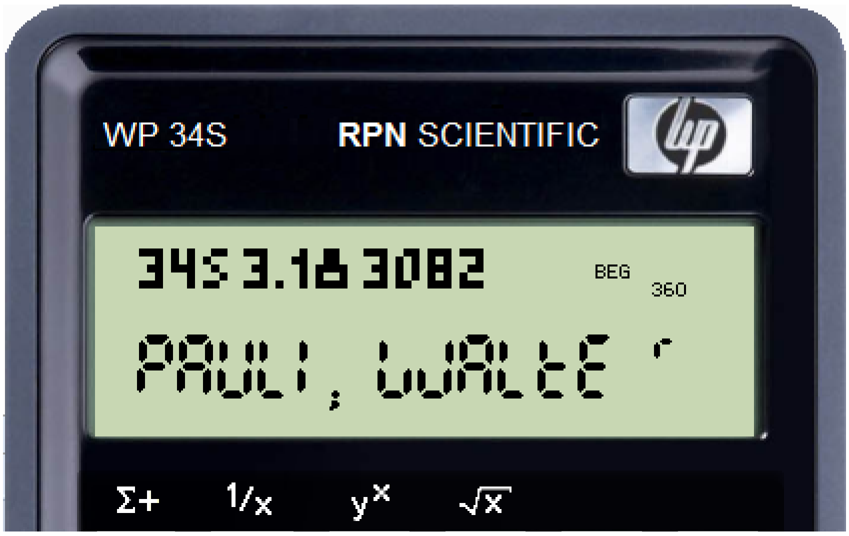
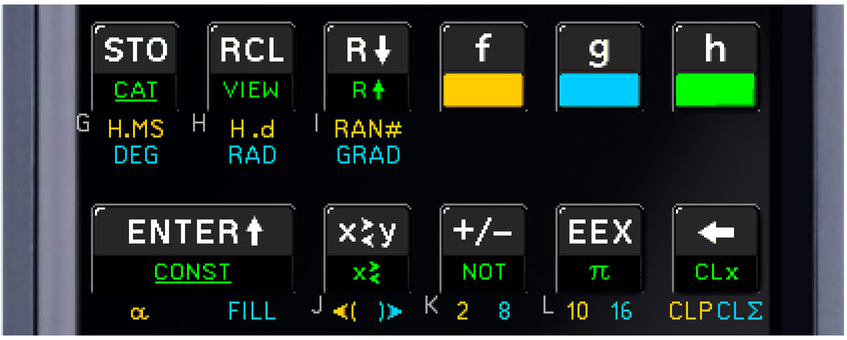


OWNER'S MANUAL



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 by you. Here is, however, first aid for those getting caught in an unexpected or unwanted calculator mode while playing before reading: H.d (i.e. f + RCL) will bring you back to default floating point mode.

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

This a preliminary manual. It may change without prior notice if the underlying software will be modified. Based on our experience in *WP 34S*, this occurred frequently so far – even basic modifications may happen. We recommend you keep watching 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.

TABLE OF CONTENTS

This project would not have reached its present state without our love for Classics, Woodstocks, Spices, Nuts, Voyagers, and Pioneers. Thus we want to quote the following statement, printed in Hewlett-Packard 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 some instructions how to convert it into a full fledge *WP 34S* yourself. Alternatively, 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 may purchase a *HP-30b*-based *WP 34S* readily in the internet, e.g. here: *http://commerce.hpcalc.org/34s.php* or here: *http://www.thecalculatorstore.com/epages/eb9376.sf/en_GB/?ObjectPath=/Shops/eb9376/Products/%22WP34s%20Pack%22* .

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

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

WELCOME!

Dear user, now you have got it: your own *WP 34S*. It uses the mechanics and hardware of a *HP-20b Business Consultant* or a *HP-30b Business Professional*, so you benefit from their excellent processor speed. And with a *HP-30b* you 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 <u>scientific</u> calculator like you have never had before** – keystroke programmable and comfortably fitting in your shirt pocket.

The function set of your *WP 34S* is based on the famous *HP-42S RPN Scientific*, the most powerful programmable *RPN* calculator built so far[1]. We expanded this set, incorporating the functions of the renowned computer scientist's *HP-16C*, fraction mode of *HP-32SII*, probability distributions like featured by *HP-21S*, and 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 as well as Chebyshev's, Hermite's, Laguerre's and Legendre's orthogonal polynomials – no more need for heavy table books,
- many statistical distributions and their inverses like Poisson, Binomial, Geometric as well as Cauchy-Lorentz, Exponential, Logistic, Weibull for reliability analysis, Lognormal and Gaussian with arbitrary means and standard deviations,
- programmable sums and products, first and second derivatives,
- testing for primality,
- extended date and time operations and a stopwatch[2] based on a real time clock,
- integer computing in fifteen bases from binary to hexadecimal,
- financial operations like mean rate of return and margin calculations,
- 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,
- 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.

---

[1] Still true, though *HP-42S* was sold in 1988 already. – Matrix operations of *HP-42S* cannot be supported by *WP 34S* for hardware reasons. A set of basic commands and several library routines cover matrices here.

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

***WP 34S* is the first *RPN* calculator overcoming the limits of a four-level stack** – forget worries about stack overflow in calculations. It features a choice of two stack sizes expanded by a complex LASTx register: traditional four stack levels for *HP* compatibility, eight levels for convenient calculations in complex domain, advanced real calculus, vector algebra 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 112 global general purpose registers, 112 global user flags, a 30 byte alpha register for message generation, up to 928 program steps in *RAM*, several thousand steps in flash memory, 16 local flags and up to 144 local registers allowing for recursive programming, and 4 programmable hotkeys for your favorite functions. Memory layout is user-settable to a large extent. And you may save your work in battery-fail-safe on-board backup memory, and communicate serially with your computer. With a little hardware upgrade, also printing on a *HP 82240A or B* is possible.

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 very beginning, we discussed our project in the *Museum of HP Calculators*, so we want to thank all the international forum members there who taught us a lot and brought their ideas and support in several stages of our project. Special thanks go to *Marcus von Cube* (Germany) 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 a *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, printing is possible thanks to gracious support by *Christoph Giesselink* (Germany), and *Ciaran Brady* (UK) wrote a *Beginners Guide* for our calculator.

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

We have carefully checked everything we could think of to our best knowledge, so our hope may be justified *WP 34S* is free of severe bugs. We cannot warrant this, however. Anyway, we promise we will continue improving *WP 34S* whenever it turns out being necessary – so if you discover any strange result, please report it to us, and if it is revealed to be an internal error we will provide you with an update as soon as we have got one ourselves. We did show short response times so far, and we will continue this way.

Enjoy!

Paul Dale and Walter Bonin

Print Conventions

- Throughout this manual, standard font is Arial. Emphasis is added by underlining or bold printing. *Specific terms, titles* or *abbreviations* are printed in italics, *hyperlinks* in blue underlined italics. Bold italic letters like ***n*** are used for variables. Calculator commands are generally called by their names, printed in capitals in running text for easy recognition. Each and every command featured is listed in the *Index of Operations* below.

- This CPX font is taken for explicit references to calculator keys. Alphanumeric and numeric displays (like Hello! and 1234) are quoted using the respective calculator fonts. `Courier` is used for file names and numeric formats.

- 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, respectively. Overall stack contents are quoted in the order [***x***, ***y***, ***z***, …] generally.

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

All this holds unless stated otherwise.

Finally: **WARNING** flags risks of severe errors. Driving your calculator in a lockup state is the worst that can happen to it as far as we know. Behave, think, and act responsibly!

GETTING STARTED

If you know how to deal with a good old *Hewlett-Packard RPN* scientific calculator, you can start with your *WP 34S* almost right away. Use the following to get information about some basic design concepts of your *WP 34S* putting it ahead of previous *RPN* scientific calculators. Continue using it as a reference manual.

> On the other hand, if your *WP 34S* is your first *RPN* scientific you use after 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*.
>
> Please read Part 1 of said manual as a starter. This part includes an excellent introduction to *RPN* – a very effective and coherent method for calculations. *RPN* is the reason why your *WP 34S* works like a charm without an [=] key. Once you got used to it you will most probably never employ a calculator featuring [=] again.
>
> Part 2 of said manual will help you when you are heading for programming your *WP 34S* for quick and easy handling of repeated or iterative computations. Further documentation, also about other calculators mentioned in the following, will add valuable information – it is all readily accessible on a single DVD from said source.
>
> Alternatively, you can get a dedicated *WP 34S Beginners Guide*, recently written by one of our users, on our website. This is the address for downloading it:
> *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 *HP-42S*. This little manual here is meant as a supplement presenting you all the new features. It contains the necessary information including some formulas and technical explanations but is not intended to replace textbooks about mathematics, statistics, physics, engineering, programming, or the like, nor 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 – though a very powerful one – it cannot think for you nor can it check the sensibility of the problem you apply it to. Do not blame us nor your *WP 34S* for errors you have made. Gather information, think before keying in and check your results: this will remain your responsibility always.

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

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 you have stored. To conserve battery energy, your *WP 34S* shuts down some five minutes after you stopped using it – when you turn it on again, you can resume working right where you left off.

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

The most striking difference to the *HP-42S* is the colorful keyboard of your *WP 34S.* Most keys feature five functions each. White print is for the *primary* function of the respective key, colored print for *secondary* functions: Green labels are put on the slanted lower faces of 34 keys, golden and blue labels are printed below of 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

.

Now let us assume you want to fence a little patch of land, 40 *yards* long and 30 *yards* wide [3], rectangular shape You have set the first corner post (A) already, and also the second (B) in a distance of 40 *yards* from A. Where do you place the third post (C) to be sure setting up the fence forming a proper rectangle? Simply key in [4]

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

So, just take a 80 *yards* rope, nail its one end on post A and its other end on B, fetch the loose loop and walk 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 care for the land, posts, rope, hammer and nails. And it will be up to you to set the posts!

In this example, pressing →P calls the function →POL. Most of the 168 labels printed on your *WP 34S* point to functions carrying simply the same name – there are only ten cases needing a little extra explanation like →P. Let us introduce them to you, starting top left on the keyboard:

---

[3] 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.

[4] 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, the „comma people" seem to be more capable to read radix points and interpret them correctly than vice versa.

1. A, B, C, and D are named *hotkeys*, since they 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 $\sqrt{x}$, 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. In analogy, SIN^{-1} stands for ASIN, etc.

3. → is a prefix for directly converting x, the value currently displayed. It may be trailed by H.MS, H.d, DEG, RAD, or GRAD (the respective function names read like e.g. →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 default floating point mode and inserts an exclamation mark in alpha mode. Cy,x and Py,x work like e.g. on *HP-15C* (calling COMB and PERM here, respectively).

9. |x| calls ABS, and RND works like e.g. on *HP-15C* (calls ROUND here).

10. There are three toggles: ./, for radix marks, P/R for programming (like e.g. on *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* below for your reference.

Let us return to our introductory example[5] for three remarks:

1. Please 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. If you want to convert results from one unit to another for any reason whatsoever, see the catalog CONV described further below.

3. Although we entered integer numbers only for both sides of our little ground, the calculation was executed in default floating point mode of your *WP 34S*. This 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 briefly introduce them to you further *below*). Before, let us show you some more typical calculations.

Calculating in Real Domain

Most of the commands your *WP 34S* features are mathematical operations or functions working with 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**,

key in	. 4 9	0.49	
and press	√x		
This returns		0.7	since $0.7^2 = 0.49$

Generally, such functions replace x (the value displayed) by the function result **f(*x*)** . The vast majority of calculators operates this way, so this is no real surprise.

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

---

[5] Generally, we assume you have finished US High School at minimum, passed Abitur, Matura or equivalent. 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 application examples was 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 of vintage *Hewlett-Packard* calculators from their very first, the *HP-9100A* of 1968, on. Be assured that almost every calculation described there for any scientific calculator can be done on your *WP 34S* significantly faster – and often even in a more elegant way.

Example: Assume having an account of 1,234 $ and taking 56.7 $ away from it. What will remain? One easy way to solve such a task works as follows:

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

That is the essence of RPN:

Enter 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[6]. Period.

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.

Stack Mechanics

Think of the stack like a pile of registers (as pictured below): bottom up, they are named **X, Y, Z, T** for tradition, 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*. ENTER↑ terminates numeric input and copies ***x*** into **Y**[7], so **X** can take another input then without losing information. Having completed that second input, − subtracts ***x*** from ***y*** and puts the result **f(*x*, *y*)** = ***y*** – ***x*** in **X** for display. This method applies for most two-number real functions.

A large fraction of mathematics can be 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*:

[6] 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* exaggerates that underlying *postfix* principle crossing 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 sticking to *RPN* with the *WP 34S*.

[7] 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 the page after next page for detailed pictures.

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

So the first parenthesis was solved as shown above. Now proceed to the second:

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	7 8 . 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 – in fact it is worth mentioning when automatic stack lift is disabled since this is under fixed conditions only and occurs far rarer.

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 [8]	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: 10.693 453 464 8 .

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 – please turn to the vintage manuals contained on the *DVD* mentioned above.

For the first time ever in a calculator, your *WP 34S* offers a choice of four or eight stack levels. Thus, the fate of stack contents depend 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. Please turn overleaf for details.

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

	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 obsolete 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 [×] [+]	$\{sin\ [\ ...)^{3/7}\}$
[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 [$\sqrt{x}$] [−] [x^2]	$\left(\sqrt{5.1}-6/5\right)^2$
[−] [|x|] .3 [y^x] 1 [+]	complete numerator
[x⇆y] [/] [$\sqrt{x}$]	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 formulas from inside out stays a wise strategy. With eight levels, however, stack overflow will hardly ever happen, even with the most advanced formulas you compute in your life as a scientist or engineer. Let your *WP 34S* do the arithmetic while you do the mathematic!

Error recovery even in long calculations is most easy in *RPN* since supported by LASTx: the special register **L** is loaded with ***x*** automatically every time just before a function is executed.

If you have erroneously executed 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 the 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 watching an attractive fellow student or collaborator you pressed [×] accidentally instead of [/] in the second last step of the example on previous page. 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 [/] [√x] (as you would have done without that mistake) and you will get the correct complete result [9].

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.

---

[9] 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 the 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 where you simply do not have to bother about such losses except in extremely rare special cases.

Calling Commands

Your *WP 34S* features significantly more than 600 different commands. Less than 170 labels are printed on the keyboard. So how do you learn about the other commands? And when you know their names, how can you access them?

The answer to the first question is easy: read! The index of operations 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.

Now assume you want to call DECOMP – it was mentioned in previous paragraph. You went to the table of contents above, looked for the *Index of Operations*, and then jumped directly to the letter **D** therein to get the necessary information what DECOMP does. OK, enter 0.375 and let it work – calling this 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	⇕ $\sqrt[3]{x}$	being the first command stored in **X.FCN**.
D	⇕ DATE	for obvious reasons.
▼	⇕ DATE→	
▼	⇕ DAY	
▼	⇕ DAYS+	
▼	⇕ DECOMP	voilà!
XEQ	y/x = 8. (360 RPN)	
x⇄y	3. (360 RPN)	since 0.375 = 3 / 8 . Expected that. But …
0.46875		??
X.FCN	⇕ DECOMP	oh, good, that function is memorized !
XEQ	y/x = 32. (360 RPN)	
x⇄y	15. (360 RPN)	meaning 0.46875 = 15 / 32 .

There is a more elegant method for calling cataloged commands described below, but this here will do for the time being.

Some Special Real Functions: Statistical Distributions, Probabilities etc.

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

- Discrete statistical distributions (e.g. Poisson, Binomial) are confined to integers. Whenever your *WP 34S* sums up a probability mass function (*pmf* [10]) $p(n)$ 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

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

- 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, **L**og**N**or**m**al, **Logis**tic, **Norm**al, **Poiss**on, Student's **t**, and **Weib**ull distributions. Chisquare and Standard Normal (Gaussian) distributions are named differently. Please see the *index* and the *catalog PROB*.

---

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

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

WARNING: This is a very coarse sketch of this topic only – please turn to 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".

There is also a wealth of commands for sample and population statistics in one and two dimensions featured. Please see the *index* and the *catalogs STAT and SUMS*. Please take a short look to two applications.

For **example**, calculating confidence limits for the 'true value' based on sample analysis, employing a particular confidence level (e.g. 95%), you must know your objective:

- Do you want to know the upper limit, below of which the 'true value' will lie with a probability of 95%? Then take 0.95 as the argument of the *qf* to get said limit – but remember there is an inevitable chance of $100\% - 95\% = 5\%$ for the 'true value' being 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 $5\% / 2 = 2.5\%$ for said value being less than the lower limit and an equal chance for it being greater than the upper limit. So you shall take 0.025 and 0.975 as arguments in two subsequent calculations using the *qf* to get both limits below and above of the sample result.

  Once again: these chances [11] 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.

Another **application**: 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 [12] :

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

STAT S ▼	÷ SERR	returns both standard errors;
XEQ		
x^2 x⇄y x^2 + √x		this is the complete denominator now;
x̄ − |x| x⇄y / STO D		and this is d.

[11] 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*".

[12] This test goes back to *DGQ* (*Deutsche Gesellschaft für Qualität*). It assumes your data are drawn from a Gaussian process, which is a frequent 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.

Also store the *degrees of freedom*:

SUMS	⇕ nΣ	recall the number of specimens
XEQ		
1 − STO J		

(learn more about the registers of your *WP 34S* in [next chapter]).

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:

. 9 7 5	0.975	
PROB U ▲	⇕ t-1(p)	as mentioned above, the requested *qf* lives in catalog PROB.
XEQ		execute 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.

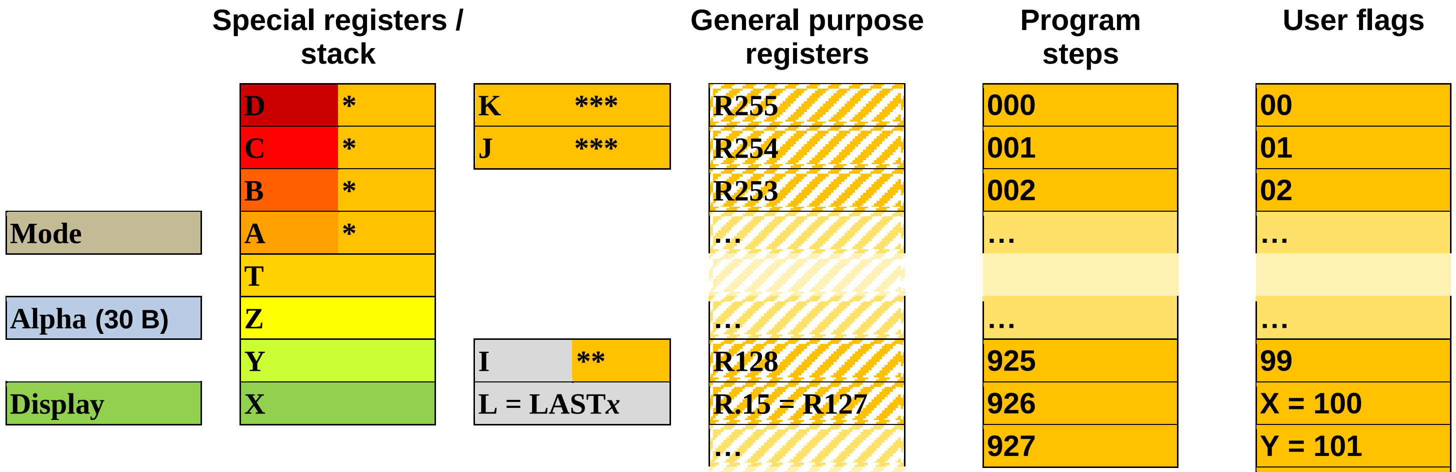
3. Let your *WP 34S* compute a new critical limit t_{cs} for f and 99.5%:

. 9 9 5	0.995	
PROB	⇕ t-1(p)	note the function is displayed immediately again when opening the catalog.
XEQ		get t_{cs}.

If $d \geq t_{cs}$ then the test indicates a *significant difference* between both samples. Else you cannot decide based on the information provided by you – your samples may contain too little data or your measurements were not precise enough.

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.

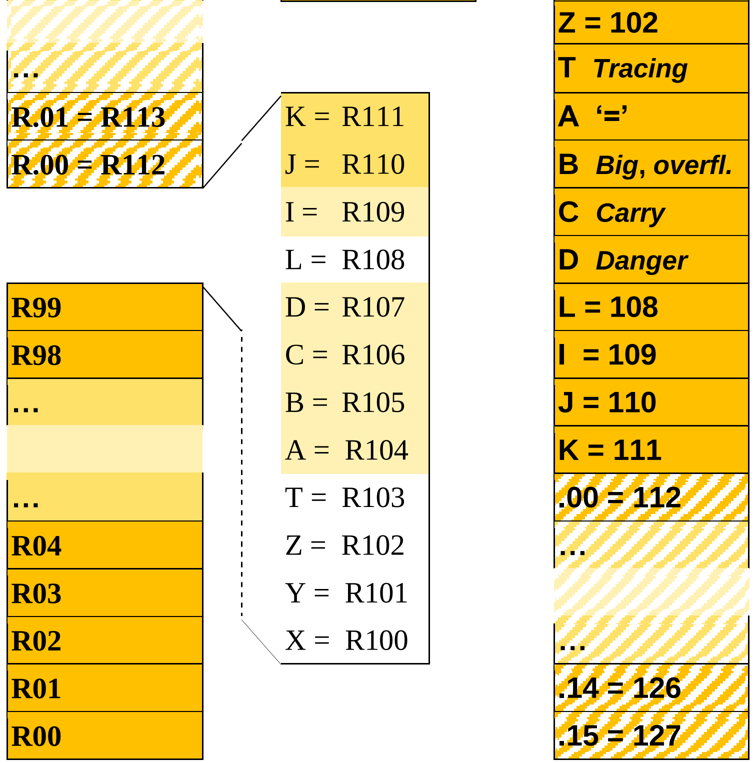
Offering two stack sizes, either **T** or **D** will be the top stack level. **A - D** will be allocated for the stack if required. **I** will be loaded in addition to **L** in complex calculations (see below). Registers **J** and **K** may carry parameters of statistical distributions. Unless required for the purposes mentioned, **A, B, C, D, I, J,** and **K** are available as additional general purpose registers.

Direct numeric addressing works up to **R99**. For indirect addressing, the lettered registers and flags carry the 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 *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 this course, the keyboard will be reassigned to work like shown here:

This kind of picture is called a *virtual keyboard.* Thereon, dark red background is used to highlight changed key functionality. White print denotes primary functions also on a virtual keyboard, 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 leaving α_T mode by EXIT.

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↑ [13] 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 number* 0 0 … 9 9, . 0 0 … . 1 5, if the respective registers are allocated. **OP? *nn*** e.g. x≠? 23	See next page for more about indirect addressing.
				Compares ***x*** with the number stored in **R23**.	

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

1	User input	[RCL], [STO], RCLS, STOS, [αRCL], [αSTO], [VIEW], [VWα+], [x≷], [DSE], [ISG], DSL, DSZ, ISE, ISZ, KEY?, [ALL], [FIX], [SCI], [ENG], DISP, BASE, bit or flag commands, etc.		
	DMD	**OP _** (with α_T mode set), e.g. RCL _ [14]		
2	User input	*Stack level or lettered register* [15] [X], [Y], [Z], .. , [K]	***Number** of register or flag or bit(s) or decimals* (see below for valid ranges)	[→] opens indirect addressing
	DMD	**OP *x*** e.g. SF K	**OP *nn*** e.g. SF 15	**OP→ _**
3	User input	Sets flag 111.	*Stack level or lettered register* [X], [Y], [Z], ... , [K]	*Register number* [0][0] … [9][9], [.][0][0] … [.][1][5], if the respective registers are allocated.
	DMD		**OP→ *x*** e.g. VIEW→L	**OP→ *nn*** e.g. STO→45
			Shows the content of the register where **L** is pointing to.	Stores ***x*** into the location where **R45** is pointing to.

Type	Number range[16] (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	

[14] 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.

[15] Exceptions: [ALL], [FIX], [SCI], [ENG], DISP, BASE don't accept lettered registers but in indirect addressing. 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 here, e.g. [STO] [+] [ENTER↑] [Z] for the latter.

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

Advanced Calculations: Real Matrices and Vectors

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

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

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

rr being the number of rows and

cc the number of columns it features. Thus the matrix has ***rr*** × ***cc*** elements.

These elements are stored in consecutive registers starting at base address |***bb***| .

> **Example:** A descriptor 7.0203 represents a 2×3 matrix – let us call it (*M*). As you know, its six elements are arranged in two rows and three columns, and 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:
>
> $\boldsymbol{m_{11} = r07 \, , \, m_{12} = r08 \, , \, m_{13} = r09 \, , \, m_{21} = r10 \, , \, m_{22} = r11}$, and $\boldsymbol{m_{23} = r12}$ **.**
>
> Depending on the current contents of these registers, the actual matrix may look like this:
>
> $$(M) = \begin{pmatrix} 2.3 & 0 & 7.1 \\ 0.4 & 8.5 & 6.9 \end{pmatrix}$$, for example.

If ***cc*** is omitted in a descriptor, it is set to ***rr*** so a square matrix is assumed. E.g. a descriptor 13.04 belongs to a 4×4 matrix with its elements stored in **R13** through **R28**. The maximum number of matrix elements is 100 – it is the number of general purpose registers available for such tasks.

Please see the *index* 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 fledge 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), where also vector multiplications are possible 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 more complicated items than real numbers. The next step are complex numbers. If you do not know them, leave them aside – you can use your *WP 34S* perfectly without them.

Else please note your *WP 34S* supports many operations in complex domain as well. The key CPX is employed as a prefix for calling complex functions. E.g. CPX f COS calls the complex cosine, and it is displayed and listed as ^{C}COS (the elevated C is the signature for complex functions 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: 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 revert 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 $x_c = x + i \cdot y$.
- … one register, e.g. **R12**, then ^{C}**f** will operate on **R12** and **R13**.
- … x and y, then ^{C}**f** will operate on x, y, z and t .

Where **one-number** <u>real</u> functions replace x by the result **f(*x*)** , one-argument <u>complex</u> 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 <u>real</u> functions replace x by the result **f(*x, y*)** as shown above. In analogy, two-argument <u>complex</u> 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}||. Please turn to the stack diagrams on next page for further details.

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.

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

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

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

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

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

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

After pressing the prefix CPX, the virtual keyboard of your *WP 34S* is open for the complex operations shown left.

Exception: STOPW is an application in real domain shown *below*.

Note that a second CPX, a ⬅, or an EXIT will only undo the

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 like this:

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.

The real number π, for **example**, may be loaded this way:
0 ENTER↑ π or via CPX π – both will result in $y = 0$ and $x = \pi$.

On the other hand, pure imaginary constants, identified by a zero real part, are entered:

imaginary_constant ENTER↑ 0 .

The complex unit ***i*** , for **example**, may be loaded this way:
1 ENTER↑ 0 or via CPX . – both will result in $y = 1$ and $x = 0$.

Compare to the stack mechanics shown above.

Addressing Complex Numbers

<table>
<tr><td>1</td><td>User input

Dot matrix display</td><td colspan="4">CPX x=? or x≠?

OP _
(with α_T mode set)
e.g.
x=_?</td></tr>
<tr><td>2</td><td>User input

DMD</td><td>0 or 1

OP *n* ?
e.g.
x=0?</td><td>*Stack level or lettered register*
X, Z, A, C, L, or J

OP? *x*
e.g.
x≠? Z</td><td>ENTER↑ [17]
leaves α_T mode

OP? _</td><td>→
opens indirect addressing.

OP?→_</td></tr>
<tr><td>3</td><td>User input

DMD</td><td>Compares $x + i\,y$ with the real number **0**.</td><td>Compares $x + i\,y$ with $z + i\,t$.</td><td>*Register number*
0 0 … 9 8,
. 0 0 …
. 1 4,
if the respective registers are allocated

OP? *nn*
e.g.
x≠? 26

Compares $x + i\,y$ with $r26 + i\,r27$.</td><td>See next page for more about indirect addressing.</td></tr>
</table>

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

1	User input	CPX RCL, STO, or x≷		
	Dot matrix display	**OP _** (with α_T mode set) e.g. ^{C}RCL _ [18]		
2	User input	*Stack level or lettered register* Z [19], A, C, L, or J	*Register number* 0 0 .. 9 8, . 0 0 … . 1 4, if the respective registers are allocated.	→ opens indirect addressing, generally working as *in real domain*.
	DMD	**OP *x*** e.g. ^{C}RCL L	**OP *nn*** e.g. ^{C}STO 18	**OP→_**
3	User input	This is ^{C}LASTx – 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	*Register number* 0 0 … 9 9, . 0 0 … . 1 5, if the respective registers are allocated.
	DMD		**OP→ *x*** e.g. ^{C}x⇆→Z	**OP→ *nn*** e.g. ^{C}STO→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: Take care of pairs, since a complex operation will always affect two registers: the one specified and the one following this. We strongly recommend storing complex numbers with their real parts at <u>even</u> register numbers always.

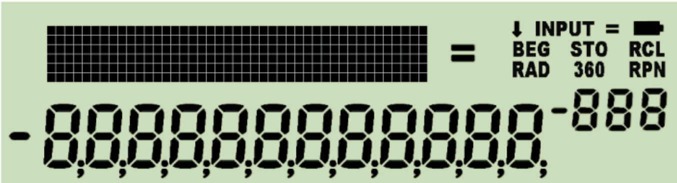
---

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

[19] Exceptions: ^{C}RCL Z, ^{C}RCL+ Z, ^{C}STO Z, and ^{C}STO+ Z require an ENTER↑ preceding Z, e.g. CPX STO + ENTER↑ Z for the latter.

DISPLAY AND MODES

The display of your *WP 34S* features three sections: numeric, dot matrix and fixed symbols. The numeric section features a minus sign and 12 digits for the mantissa, as well as a minus sign and 3 digits for the exponent. The dot matrix is 6 dots high and 43 dots wide, allowing for some 7 to 12 characters, depending on their widths. The fixed symbols on the top right side (except the big '=') are called *annunciators*, and are for indicating modes.



The dot matrix section above is used for

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

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

If two or more requests concur for display space, the priorities are as follows:

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

Modes and Annunciators

The annunciators or specific characters in the 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*		Serial I/O in progress (see App. A).	
▬	battery low (≤ 2.5 *V*)	battery voltage > 2.5 *V*	*WP 34S* will shut off automatically when voltage drops below 2.1*V*.	

Indicator	Set by	Cleared by	Explanation, remarks	Sets mode
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
4__	BASE 4			4
5__	BASE 5			5
6__	BASE 6			6
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	11
-2__	BASE 12			12
-3__	BASE 13			13
-4__	BASE 14			14
-5__	BASE 15			15
h__	16		Hexadecimal integer mode	h
_c_	carry, SF C	CF C	Indicate the respective bits set in integer modes (see *below*)	
__o	overflow, SF B	CF B		
C	complex result	else	Indicates a complex result returned by the last operation (see *above*)	
D	DBLON	DBLOFF	See *Appendix E*	
G	GRAD	DEG, RAD	Angular mode (see next page)	
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

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. Please check the examples shown below.

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

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

> **WARNING:** Note the user is responsible for recalling valid mode data – else your *WP 34S* may be driven into a lockup state! See the *index of operations* for more information about changing modes and the individual commands employed.

Some regional formatting preferences may be set at once using shortcuts:

Command	Radix mark[20]	Time	Date[21]	JG [22]	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 also to South America.
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	

[20] 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.

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

[22] 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.

Also the angular modes deserve a closer look: there are three of them, DEG, RAD, and GRAD [23]. And degrees (DEG) may be displayed in decimal numbers as well as in hours, minutes, seconds and hundredth of seconds (H.MS). Conversions are provided for going from one to the other:

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

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

Command and Mode Specific Output in Real (and Complex) Domain

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 per 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*.

Some commands and modes use the display in a special way. The respective operations are listed below. A large fraction of them present *temporary messages* [24].

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

[24] 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.

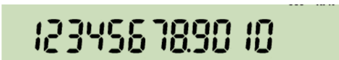
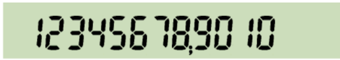
1. For **floating point decimal numbers**, the mantissa will be displayed adjusted to the right, the exponent to the left. Within the mantissa, either points or commas may be selected as radix marks, and additional marks may be chosen to separate thousands.

   Assume the display set to FIX 4. Key in 1 2 3 4 5 6 7 8 . 9 0 1 ENTER↑, and you will get

    or 

   with thousands separators on [25].

   Without them, the same number will look like this:

    or 

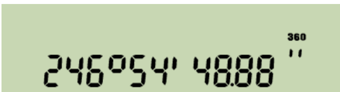
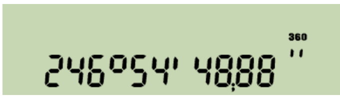
   With ENG 3 and after +/-, you will get

    or 

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

   Floating point decimal numbers within $10^{-383} \leq |x| < 10^{+385}$ may be entered directly easily. Within this range, your *WP 34S* calculates with 16 digits [26]. Smaller values than 10^{-398} are set to zero. For results $|x| \geq 10^{+385}$, error 4 or 5 will appear (see *below*).

2. 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 e.g.

    or ,

   depending on the radix setting.

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

   .

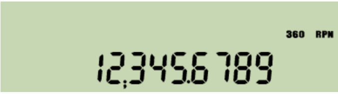
If such data are displayed outside of a catalog or browser (see a separate chapter about these below), they will vanish with the next keystroke. EXIT or ◄ will just clear the extra data returning to the normal display, any other key will be executed.

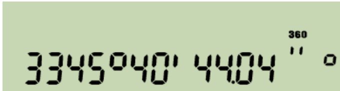
[25] These separators may also be beneficial in fraction mode described below.

[26] 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}. Divide it by 2 instead and the result will become zero. See Appendix B for the reasons.

Note there are no leading zeroes, neither in the hours nor minutes nor seconds sections.

For times or angles exceeding 9,000, an o will be shown there signaling an overflow, and the value is displayed modulo 9,000. For **example**:



will become after H.MS :


until the next key is pressed.

3. **Fitting** a number of data points with a regression curve, four different mathematical models are provided as in *HP-42S*. See the commands EXPF, LINF, LOGF, POWERF in the *index* below. The command BESTF will set your *WP 34S* to select the model resulting in the greatest absolute correlation coefficient automatically (see CORR).

   As shown in this **example**

,

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

4. ◀ and ▶ in DECM show the full mantissa of x, i.e. all digits present internally, and the exponent as a temporary message. For **example**, π ◀ returns

.

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

   Example: Enter the following:

   0 h MODE D ▼ ▼ ▼ DENMAX [27]

   XEQ

   4 7 . 4 0 6 2 5 +/- a b/c

[27] This is picking a function from a catalog. The general procedure for doing this is explained further below.

and you will see -47 13/32

or after d/c - 15 17/32 [28].

Squaring this improper fraction results in 230 1289/1024.

Now, enter a b/c for converting this into a *proper* fraction [29]. You get

⌞47 361/1024

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 ▶ to unveil the integer part of this proper fraction is <u>22</u>47.

Input in fraction mode is straightforward and logically coherent:

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

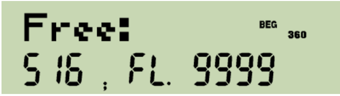
For comparison, note *HP-32SII* reads the last input here as ½ – which is, however, not consistent with its other input interpretations in fraction mode.

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

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

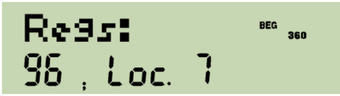
Further Commands Returning Specific Displays

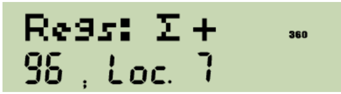
1. STATUS returns very useful information about current memory allocation and the space available. It shows the amount of free memory words in *RAM* and flash first, e.g.:



.

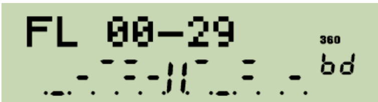
Press ▼ and read if there are summation registers used, plus the number of global numbered registers and local registers allocated (note these are emulator displays):



or


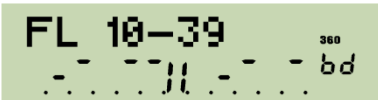
.

Another ▼ will present the status of the first 30 user flags, shown very concisely in one display, allowing an immediate status overview after some training. For **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 displayed usually, are three rows of horizontal bars now. Each shows the status of 10 flags. With a flag set, the respective 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. Following top row bars indicate flag 5 set, 6 clear, and 7 set. Next two rows show the status up to flag 29 as expected.

Pressing ▼ once will proceed to displaying flags 10 - 39 in the same format:



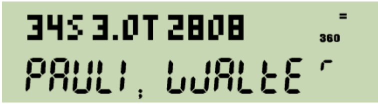
Another ▼ shows flags 20 - 49 etc. until 70 - 99, 80 - 99, 90 - 99. A final ▼ displays the last 12 global flags in rows of four – note flag X is shown being set as we expect:

▲ will browse backwards.

Alternatively, pressing a digit (e.g. 5) will display up to 30 flags starting with 10 times this digit (e.g. flags 50 – 79 here). Pressing a legal letter like D will display the top 12 flags. The numeric exponent always indicates the status of the four hotkeys top left on the keyboard – if all four labels are defined in programs then ALL will be shown there.

The status will be displayed this way until EXIT or ← is pressed.

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 and capacitors are built in and the timer firmware is installed in your *WP 34S*, 'T' will trail the version number as displayed here:

   .

   If also the IR-diode is built in 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.

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

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

   .

   Expect similar displays after DAYS+.

   Dates before the year 8 may be indicated differently to what they really were due to the inconsistent application of the leap year rule before this. We count on your understanding.

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

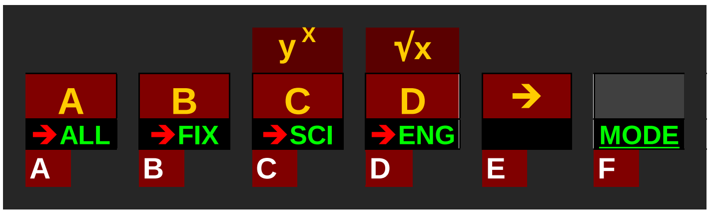
SOMETHING DIFFERENT: INTEGER MODES

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 *below*).

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 $\sqrt{x}$ here [30]). To ease operation in different modes, pressing any key (or a sequence of prefix(es) and a key) will display its present assignment in the top line for checking. Holding down the last key 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.

[30] 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 – a loss not hurting us anyway here. Reassignments are generally chosen this way. – Please 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**. 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 *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**:

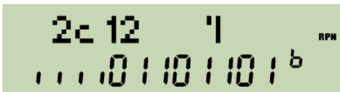
Set your *WP 34S* to WSIZE 12, LZON. This setting allows seeing all 12 bits in one calculator display easily. Enter 147. Then switch to 1COMPL, BASE 2. You will see:

and – after +/- –


.

Please forget the 'l 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. As you see, +/- in **1c** inverts every single 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 just flips the top bit in **sm**, that's all.

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.

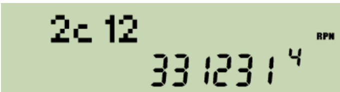
This needs explanation, since changing signs has no meaning in **un** per definition [31], where the most significant bit adds magnitude, not sign, so the largest value represented by a 12-bit word is 4095 instead of 2047. Thus, [+/-] should be illegal here and result in no operation. Nevertheless, [+/-] was allowed and implemented in *HP-16C*, so we follow this implementation for sake of backward compatibility.

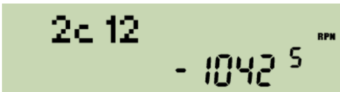
Thus, pressing [+/-] will not suffice anymore for returning to the original number here, you must also clear the overflow flag by [CF] [B] explicitly (see *above*).

As you have seen, positive numbers stay unchanged in all those modes. Just negative numbers 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. -147 d in default **2c** will be displayed as -146 d in **1c**, -1901 d in **sm**, and 3949 d 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.

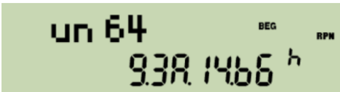
Compare for **example** the outputs

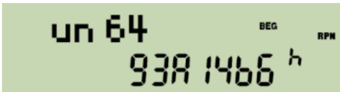


for base 4 and


for base 5 [32].

Let us look to bigger words now: The following **example** shows your *WP 34S* displaying an arbitrary number in unsigned hexadecimal mode with word size 64, with or without separators:



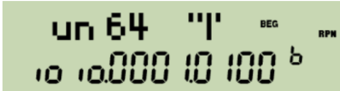
or


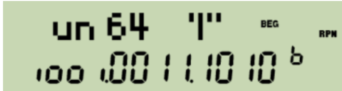
After switching to binary mode, this number will need 28 digits, being 1001001110100001010010110110. The 12 least significant digits are displayed initially together with an indication that there are four display windows in total with the rightmost shown:

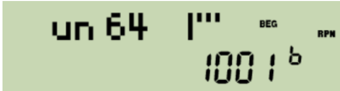
The least significant byte is emphasized. Press [◄] and you will get the next bytes (note there is a 4-bit overlap with the previous display):

[31] 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.

[32] 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.



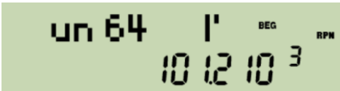




The last display shows the four most significant bits of this binary number. If leading zeros were turned on, there will be eight display windows (corresponding to eight bytes) here, with the four 'most significant' containing only zeros.

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

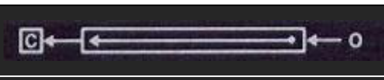
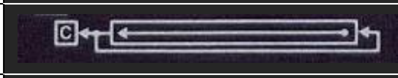
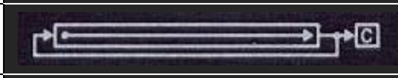
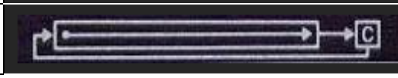
Browsing a large integer in steps of eight digits is a specialty of binary mode. In any other base the step size is the full display width, i.e. twelve digits without any overlap. See e.g. the most and least significant parts of the same number in base 3:



.

Bitwise Integer Operations

Your *WP 34S* carries all the operations you may know from vintage *HP-16C* plus more. You will find the schematic pictures in the table below as printed on the backplate of *HP-16C*. The 'C' in a box stands for the carry bit. The following **examples** deal with 8-bit words showing leading zeros for easy reading. Input is 11101011 b generally. For further details about the respective operations, please turn to the *index* below.

	Schematic	Example (outputs in 2COMP, UNSIGN and SIGNMT)			
SL		SL 2	10101100 bc		
SR		SR 3	00011101 b		
ASR		ASR 3	11111101 b	00011101 b	10001101 b
RL		RL 2	10101111 bc		
RR		RR 3	01111101 b		
RLC		RLC 2	10101101 bc		
RRC		RRC 3	11011101 b		

The outputs in mode 1COMP are identical to those for 2COMP in these examples. Note the picture for ASR (i.e. arithmetic shift right) correctly describes this operation for 1's and

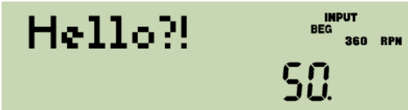
2's complement modes only. In all modes of *HP-16C*, however, ASR 3 equals a signed division by 2^3, hence the different results for the latter two modes shown above. The other bitwise operations are insensitive to complement mode setting.

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

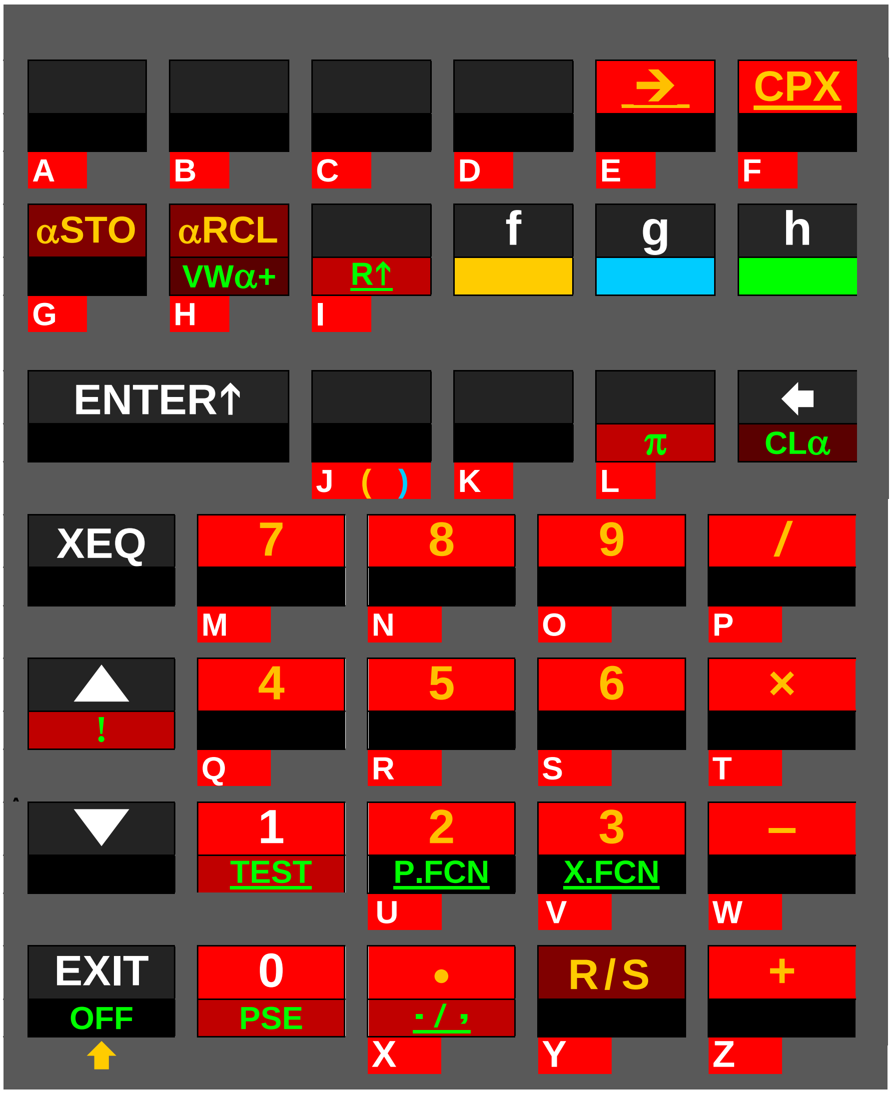
See the *index* for further commands working on bit level in integer modes (AND, OR, XOR, NOT, NAND, NOR, XNOR, LJ, RJ, MASKL, MASKR, RAN#, and nBITS).

SOMETHING COMPLETELY DIFFERENT: FULL ALPHA MODE

Alpha mode is designed for text entry, e.g. for prompts. In this mode, the alpha register is displayed in the upper part of the LCD. 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:



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 in 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*.

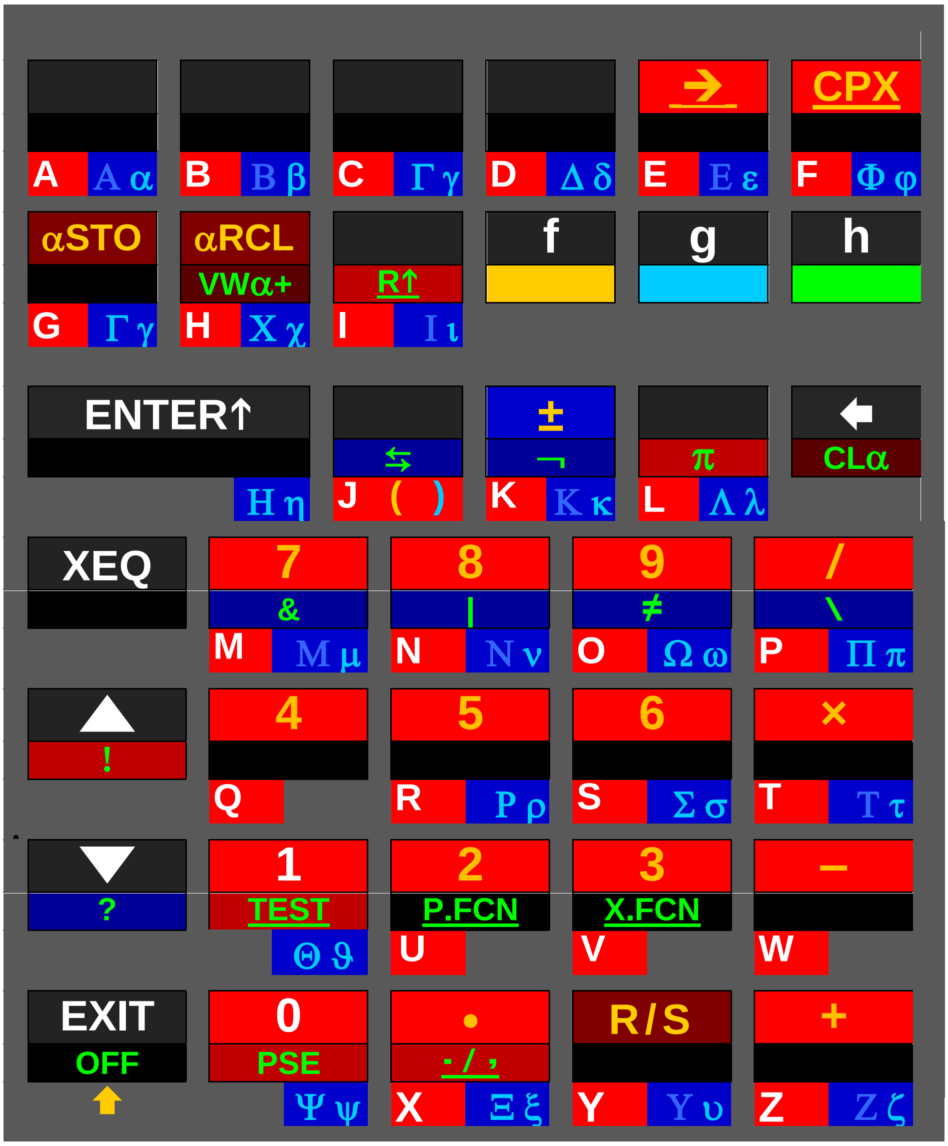
Those labels printed on darker red background changed their functionality in other ways. See the keys [STO], [RCL], and [R/S], as well as the labels [VIEW] and [CLx].

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 [33].

---

[33] 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 [34]. And h allows also accessing logic symbols via the Boolean operations.



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

For a less colorful picture of this virtual keyboard, printed in a size you can attach it to the back of your *WP 34S*, please turn to *Appendix A*.

[34] "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 since looking exactly like the Latin letter **O** in either case. Where we printed Greek capitals with lower contrast, they look like the respective Latin letters in our fonts. Greek professors, we count on your understanding.

Character Set and Fonts

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

See here all characters directly evocable through the virtual alpha keyboard (as shown above):

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

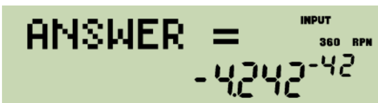
As soon as a string exceeds the visible display using the large font, your *WP 34S* will take the small font automatically to show as much as possible:

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

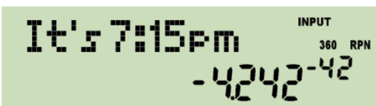
Many more characters of both fonts live in the alpha catalogs. You will find them below.

Displaying the Alpha Register

As soon as you enter alpha mode and as long as you stay therein, ***alpha*** is 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 index of operations below. For **example**, αTIME allows creating texts like

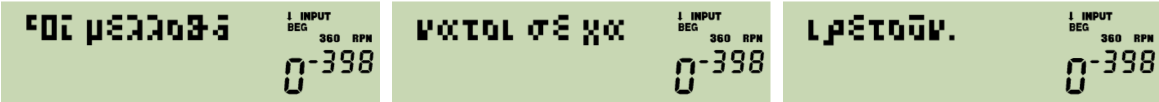


or

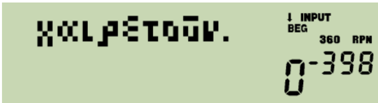
depending on time mode setting (12h / 24h).

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

Please 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 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 for this – it will show you the left end (i.e. the first characters ***alpha*** is containing).

Nevertheless we will not forget your *WP 34S* is mainly paid for working as a calculator.

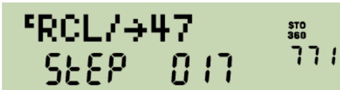
PROGRAMMING

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

The basic building blocks within program memory are routines (or programs). Typically, a routine starts with a LBL statement and ends with RTN or END. In between, you may store any sequence of instructions (commands, operations, statements) for repeated use. Choose any operation featured – only a few commands are not programmable. The statements in your routine may use each and every register provided – there are (almost) no limits. You are the sole and undisputed master of the memory!

This freedom, however, has a price: you shall take care that your routines do not interfere in their quest for data storage space. So it is good practice recording the registers used by a particular routine, and documenting 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 program-specific step counting like it was in *HP-42S*.

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.

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

1. If ***xy*** is purely numeric or a hotkey, it will be searched forward from the current position of the program pointer. When an END statement is reached without finding ***xy***, the quest will continue right after previous END (so the search will stay in the current routine). This is the search procedure for local labels. It is as known from *HP-41C*.

2. Else, i.e. if ***xy*** is an alpha label of up to three characters of arbitrary case (automatically enclosed in ' like **'Ab1'**), searching will start at program step 000 and cover the entire memory in the order *RAM*, *FM*, and *XROM*, independent of the position of the program pointer. This is the search procedure for global labels.

Addressing Labels

1 User input	A, B, C, or D	XEQ, GTO, LBL, LBL?, SLV, ∫f, 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.	→ [35] opens indirect addressing and sets α_T mode.		*2-digit numeric (local) label* 0 0 … 9 9
Dot matrix display		**OP *label*** e.g. Σ B	**OP '_**	**OP →_**		**OP *nn*** e.g. LBL 07
3 User input		Sums up the function given in a routine labeled **B**.	***Alphanumeric (global) label*** (1 to 3 characters [36])	*Stack level or lettered register* X, Y, Z, … , K	*Register number* 0 0 … 9 9, . 0 0 … . 1 5, if applicable [37]	
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 *index of operations* for special cases applying to this command exclusively.

---

[35] Works with all these operations except LBL .

[36] 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.

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

Tests

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

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

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

Local Data

If – after some time – you have a number of different routines stored, keeping track of their memory requests may become a challenge. Most of modern programming languages take care of this problem by declaring *local variables*, i.e. memory space allocated from general data memory and accessible for the current routine only – when the routine is finished, the respective memory is released again. On the *WP 34S*, registers are for data storage – so we offer you *local registers* allocated to your routine exclusively.

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

1. You just enter the command LOCR 5 in your routine specifying you want five local registers,
2. then you may access these registers most easily using local numbers **.**00 … **.**04 throughout *P1*.

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

This construction allows e.g. for recursive programs, since every time such a routine is called again it will get a new set of local registers being different from the ones it got before. Nevertheless, since you remain the sole and undisputed master of the memory, proper programming and care-taking persist being your job.

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

Programmed Input and Output, User Interaction and Dialogues

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

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+. Please 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 .

Please see the index 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. A typical program structure might look like this **example**:

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	
…		Compute the new number for missing numeric user input.
…	RTN	
…	LBL B	Called if user input after step 006 was terminated by B.
…		
…	RTN	
…		
…	END	

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*, it will be necessary moving the program counter (PC) into the desired program and stopping there – provided programs are separated by END.

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.

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

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

If, for **example**, you want to 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 might 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 as a row / column code in register ***nn***: 0 to 9 for the respective digits; 10 for [.], [+/-], and [EEX]; 11 for [f], [g], and [h]; and 12 for the other keys. An invalid code in the target register throws an 'Invalid Range' error.

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

Flash Memory (*FM*) and *XROM*

In addition to the *RAM* provided, your *WP 34S* allows you accessing 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 defined already – labels in *FM* are tagged with Lib there.

FM is ideal for backups or other long-living data, but shall not be used for repeated transient storage like in programmed loops [38]. Registers and standard user program memory,

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

residing in *RAM* on the opposite, are designed for data changing frequently but will not hold data with the batteries removed. So both kinds of memory have specific advantages and disadvantages you shall take into account for optimum benefit and long lasting joy with your *WP 34S*. Find more about *FM* in *Appendix A* below.

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

INDEX OF OPERATIONS

All commands available (more than 550) are found below with their *names* and the *keystrokes* necessary. Names printed in **bold** face in this list belong to functions directly accessible on the keyboard, the other commands may be picked from *catalogs*. The command names will show up identically in catalogs and program listings unless specified otherwise explicitly. Sorting in index and catalogs is case insensitive and works in the following order:

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

Super- 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 *HP-42S*, bit and integer functions as on *HP-16C*, unless stated otherwise under remarks. Please refer to the manuals of the vintage calculators mentioned for additional information about traditional commands.

An elevated '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 occurs, a capital C in the dot matrix will remind you to look at ***y*** as well.

Over 275 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. **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.

Name	Keys to press	in modes	Remarks (see above for general information)
10^x	f 10^x	¬α	(1)
12h	h MODE 12h	¬α	(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	¬α	(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	¬α	(0) Sets 24h time display mode. Compare 12h.
2COMPL	h MODE 2COMPL	¬α	(0) Sets 2's complement mode for integers.
2^x	f 2^x	¬α	(1)
$\sqrt[3]{x}$	h X.FCN $\sqrt[3]{x}$	¬α	(1)
ABS	f |x|	¬α	(1) Returns the absolute value. ^{C}ABS 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) Inverse hyperbolic cosine, known as $\operatorname{arcosh}(x)$. Note there is no need for pressing f here.
AGM	h MODE AGM	DECM	(2) Returns the arithmetic-geometric mean.
ALL	h ALL ***n***	¬α	(0) ALL 0 works almost like ALL in *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 $x < 10^{-n}$ <u>and</u> more than 12 digits are required to show x completely. **Example:** Input: 700 / ALL 03 / 1/x / 10 / ; Display: 700 / 700. / 0.00 142857 143 / 1.42857 142857 $^{-4}$
AND	h AND	Integer	(2) Works bitwise as in *HP-16C*.
		DECM	(2) Works like AND in *HP-28S*, i.e. x and y are interpreted before executing this operation. Zero is 'false', any other real number is 'true'.

Name	Keys to press	in modes	Remarks (see *above* for general information)
ANGLE	h X.FCN ANGLE	DECM	(2) Returns the angle between positive x-axis and the straight line connecting the origin and the point (x, y) , i.e. $arctan(^y/_x)$.
ASIN	g SIN⁻¹	DECM	(1) Returns $\arcsin(x)$.
ASINH	g HYP⁻¹ SIN	DECM	(1) Inverse hyperbolic sine, known as $\operatorname{arsinh}(x)$. 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 *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) Inverse hyperbolic tangent, known as $\operatorname{artanh}(x)$. Note there is no need for pressing f here.
BASE	h MODE BASE ***n***	¬α	(0) Sets the base for integer calculations, with $2 \le n \le 16$. Popular bases are directly accessible on the keyboard. See *above* for more information about the display in integer modes. Furthermore, BASE 0 sets DECM, and BASE 1 calls FRACT. See below. **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*).
BASE10	f 10		
BASE16	g 16		
BASE2	f 2		
BASE8	g 8		
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) Returns this voltage in units of 100mV.
BC?	h TEST BC? ***n***	Integer	(0) Tests the specified bit in x.
BestF	h MODE BestF	DECM	(0) Selects the best curve fit model, maximizing the correlation like BEST does in *HP-42S*.

Name	Keys to press	in modes	Remarks (see above for general information)
Binom	h PROB Binom	DECM	(1) Binomial distribution with the number of successes ***g*** in **X**, the probability of a success $\boldsymbol{p_0}$ in **J** and the sample size ***n*** in **K**. Binom_P [39] 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 $\boldsymbol{F_B}$ in **X**, $\boldsymbol{p_0}$ 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**: $B_n=(-1)^{n+1}n\cdot\zeta(1-n)$. 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).
B_n*	h X.FCN B_n*		
BS?	h TEST BS? ***n***	Integer	(0) Tests the specified bit in ***x*** .
Cauch	h PROB Cauch	DECM	(1) 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 $F_{Ca}(x)=\frac{1}{2}+\frac{1}{\pi}\arctan\left(\frac{x-x_0}{\gamma}\right)$. Cauch^{-1} returns ***x*** for a given probability $\boldsymbol{F_{Ca}}$ in **X**, $\boldsymbol{x_0}$ 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.
CLALL	h P.FCN CLALL	¬(α, PR)	(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.

[39] Binom_P equals BINOMDIST(***g; n; p₀; 0***) and Binom equals BINOMDIST(***m; n; p₀; 1***) in MS Excel.

Name	Keys to press	in modes	Remarks (see above for general information)
CLPALL	h P.FCN CLPALL	¬(α, PR)	(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 h P.FCN CLSTK	¬α	Clears all stack registers currently allocated, i.e. **X** through **T** or **X** through **D**, respectively. All other register contents are kept.
CLx	h CLx	¬α	Clears register **X** only, disabling stack lift as usual.
CLα	h CLx h P.FCN CLα	α ¬α	(0) Clears the alpha register like CLA in *HP-42S*.
CLΣ	g CLΣ	DECM	(0) Clears the summation registers and releases the memory allocated for them.
COMB	f Cy,x	¬α	(2) Returns the number of possible sets of y items taken x at a time. No item occurs more than once in a set, and different orders of the same x items are 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 y, thus returning the complex conjugate of x_c.
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 **X**.
COSH	f HYP COS	DECM	(1) Returns the hyperbolic cosine of x.
COV	h STAT COV	DECM	(-1) Returns the population covariance for two data sets. It depends on the fit model selected. For LinF, it calculates $COV_{xy} = \frac{1}{n^2}\left(n\sum x_i y_i - \sum x_i \sum y_i\right)$ See s_{XY} for the sample covariance.
ᶜCROSS	CPX h X.FCN ᶜCROSS	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 $[x \cdot t - y \cdot z$, 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 *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 $\mathbf{Y}$ 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 *HP-12C*.
DBLR	h X.FCN DBLR	Integer	(2) Double word length commands for remainder, multiplication and division like in *HP-16C*. If the division remainder is $\neq 0$, 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 $\mathbf{X}$ contains 2.25 then DECOMP will return $x = 4$ and $y = 9$, pushing previous content of $\mathbf{Y}$ to $\mathbf{Z}$ 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.
DENANY	h MODE DENANY	¬α	(0) Sets default fraction format like in *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.

Name	Keys to press	in modes	Remarks (see above for general information)
DENFAC	h MODE DENFAC	¬α	(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.
DENFIX	h MODE DENFIX	¬α	(0) Sets fixed denominator format, i.e. the one and only denominator allowed is the value set by DENMAX.
DENMAX	h MODE DENMAX	¬α	(0) Works like /c in *HP-35S*, but the maximum denominator settable is 9,999. It will be set to this value if $x < 1$ or $x > 9{,}999$ at DENMAX execution time. For $x = 1$ the current setting is recalled.
DET	h MATRIX DET	¬α	(1) Takes a descriptor of a square matrix in $\mathbf{X}$ and returns the determinant of the matrix. The matrix itself is not modified.
DISP	h MODE DISP *n*	DECM	(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).
ᶜDOT	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, …], dropping two stack levels.
DROP	h X.FCN DROP	¬α	Drops x. See above for details.
	CPX h X.FCN ᶜDROP	DECM	Drops x_c. See above for details.
DSE	f DSE *r*	PR	(0) Given `cccccc.fffii` in r, DSE decrements r by `ii`, skipping next program line if then `ccccccc` ≤ `fff`. If r features no fractional part then `fff` is 0 and `ii` is set to 1. Note that neither `fff` nor `ii` can be negative, and DSE makes only sense with `cccccc` > 0 .
DSL	h P.FCN DSL *r*	PR	(0) Works like DSE but skips if `ccccccc` < `fff` .
DSZ	h P.FCN DSZ *r*	PR	(0) Decrements r by 1, and skips the next step if $\lvert r \rvert < 1$ thereafter. Known from *HP-16C*.
D.MY	h MODE D.MY	¬α	(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…

Name	Keys to press	in modes	Remarks (see above for general information)
D→R		DECM	(1) See the catalog of conversions for conversions from degrees to radians.
E3OFF	h MODE E3OFF	¬α	(0) Toggle the thousands separators for DECM (points or commas depending on radix setting).
E3ON	h MODE E3ON		
END	h P.FCN END	PR	(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 complementary: $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2} d\tau$ and $erfc(x) = 1 - erf(x)$.
erfc	h X.FCN erfc		
ERR	h P.FCN ERR *n*	¬α	(0) Raises the error specified. See below for the respective error codes.
EVEN?	h TEST EVEN?	¬α	(0) Checks if x is integer and even.
e ^{x}	f e^x	¬α	(1)
ExpF	h MODE ExpF	DECM	(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$_P$ [40] 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}		

[40] 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)
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) Returns more accurate results for the fractional part of e^x with $x \approx 0$.
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 FC?F FC?S	h TEST FC?C ***n*** etc.	¬α	(0) Tests if the flag specified is clear. Clears, flips, or sets this flag after testing, respectively.
FF	h P.FCN FF ***n***	¬α	(0) Flips the flag specified.
FIB	h X.FCN FIB	¬α	(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 $n \geq 2$.
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 TEST 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	¬α	(0) Sets fraction mode like in *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.

Name	Keys to press	in modes	Remarks (see above for general information)
$F_P(x)$	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 *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 $F^{-1}(p)$		
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*** ± *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*** [41].
g_d	h X.FCN g_d	DECM	(1) Returns the Gudermann function or its inverse $g_d(x)=\int_0^x \frac{d\xi}{\cosh\xi}$ or $g_d^{-1}(x)=\int_0^x \frac{d\xi}{\cos\xi}$, respectively.
g_d^{-1}	h X.FCN g_d^{-1}		
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 $\boldsymbol{p_0}$ for a success in each such experiment must be specified in **J**. Geom $^{-1}$ returns the number of failures ***f*** before 1st success for given probabilities $\boldsymbol{F_{Ge}}$ in **X** and $\boldsymbol{p_0}$ in **J**.
$Geom_P$	h PROB $Geom_P$		
$Geom_u$	h PROB $Geom_u$		
$Geom^{-1}$	h PROB $Geom^{-1}$		
GRAD	g GRAD	DECM	(0) Sets angular mode to gon or grads.
GRAD→	h X.FCN GRAD→	DECM	(1) Takes ***x*** as given in gon or grads and converts them to the angular mode currently set.

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

Name	Keys to press	in modes	Remarks (see *above* for general information)
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 … (not programmable) … to one of these labels, if defined.
	h GTO . ***nnn***	¬α	… to step ***nnn*** .
	h GTO . ▲	¬α	… directly after previous END, going to the top of current program.
	h GTO . ▼	¬α	… directly after next END, going to the top of next program.
	h GTO . .	¬α	… to step 000, i.e. top of *RAM*.
GTOα	h P.FCN GTOα	¬α	(0) Takes the first three characters of ***alpha*** (or less if there are less available) as a label and positions the program pointer to it.
H_n	h X.FCN H_n	DECM	(1) Hermite's polynomials for probability: $H_n(x) = (-1)^n \cdot e^{x^2/2} \cdot \frac{d^n}{dx^n}\left(e^{-x^2/2}\right)$ with ***n*** in **Y**, solving the differential equation $f''(x) - 2x \cdot f'(x) + 2n \cdot f(x) = 0$.
H_{np}	h X.FCN H_{np}	DECM	(1) Hermite's polynomials for physics: $H_{np}(x) = (-1)^n \cdot e^{x^2} \cdot \frac{d^n}{dx^n}\left(e^{-x^2}\right)$ with ***n*** in **Y**.
H.MS	f H.MS	DECM	(1) Assumes **X** containing <u>decimal</u> hours or degrees, and displays them converted in the format `hhhh°mm′ss.dd″` as shown in the paragraph *above*. Will return to the previous decimal display with the next keystroke.
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–	DECM	

Name	Keys to press	in modes	Remarks (see *above* for general information)
IBASE?	h TEST IBASE?	¬α	(-1) Returns a number from 2 to 16 according to the integer base set (see BASE).
IMPFRC	g d/c	¬α	(1) Sets fraction mode allowing improper fractions in display (i.e. $^5/_3$ instead of $1\,^2/_3$). Converts x according to the settings by DEN… Absolute decimal equivalents of x must not exceed 100,000. Compare PROFRC.
		FRC	(1) Allows displaying improper fractions. Thus converts a proper fraction in **X** into the equivalent improper fraction, if applicable.
INC	h P.FCN INC *r*	¬α	(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*	PR	(0) Works like ISG but skips if `ccccccc` ≥ `fff` .
ISG	g ISG *r*	PR	(0) Given `cccccc.fffii` in ***r***, this function increments ***r*** by `ii`, skipping next program line if then `ccccccc` > `fff` . If ***r*** features no fractional part then `ii` is set to 1. Note that neither `fff` nor `ii` can be negative, but `cccccc` can.
ISZ	h P.FCN ISZ *r*	PR	(0) Increments ***r*** by 1, skipping next program line if then $\lvert r\rvert < 1$. Known from *HP-16C*.
I β	h X.FCN Iβ	DECM	(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.
$I\Gamma_p$	h X.FCN IΓp	DECM	(2) Returns the regularized (incomplete) gamma function $P(x,y)=\frac{\gamma(x,y)}{\Gamma(x)}$. For $\gamma(x,y)$ see γ_{XY} , for $\Gamma(x)$ see *there*.

Name	Keys to press	in modes	Remarks (see above for general information)
$I\Gamma_q$	h X.FCN IΓq	DECM	(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}, for $\Gamma(x)$ see there.
JG1582	h MODE JG1582	DECM	(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, 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 TEST 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.
LBL	f LBL ***label***	PR	(0) Identifies programs and routines for execution and branching. See opportunities for specifying ***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*** [42].
LEAP?	h TEST LEAP?	DECM	(0) Takes ***x*** as a date in the format selected, extracts the year, and tests for a leap year.

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

Name	Keys to press	in modes	Remarks (see above for general information)
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. LgNrm returns $F_{Ln}(x) = \Phi\left(\frac{\ln x - \mu}{\sigma}\right)$ with $\Phi(z)$ denoting the standard Normal cdf. LgNrm$_P$ returns $f_{Ln}(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\sigma^2}}$. LgNrm^{-1} returns ***x*** for a given probability ***F$_{Ln}$*** in **X**, ***μ*** in **J**, and ***σ*** in **K**.
LgNrm$_P$	h PROB LgNrm$_P$	DECM	
LgNrm$_u$	h PROB LgNrm$_u$	DECM	
LgNrm^{-1}	h PROB LgNrm^{-1}	DECM	
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	DECM	(0) Selects the linear curve fit model $y = a_0 + a_1 x$.
LJ	h X.FCN LJ	Integer	(-1) Left justifies a bit pattern within its word size as in *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 L$_n$	DECM	(2) Laguerre's polynomials (compare L$_n$α below) $L_n(x) = \frac{e^x}{n!} \cdot \frac{d^n}{dx^n}\left(x^n e^{-x}\right) = L_n^{(0)}(x)$ with ***n*** in **Y**, solving the differential equation $x \cdot y'' + (1-x)y' + ny = 0$.
LN1+x	h X.FCN LN1+x	DECM	(1) Natural logarithm of values close to zero. Returns $\ln(1+x)$, providing a higher accuracy in the fractional part of the result.
L$_n$α	h X.FCN L$_n$α	DECM	(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**.
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 Γ(***x***). See there.

Name	Keys to press	in modes	Remarks (see above for general information)
LOAD	h P.FCN LOAD	¬(α, PR)	Restores the entire backup from flash, i.e. executes LOADP, LOADR, LOADSS, LOADΣ, and returns Restored then. Compare SAVE. See the commands mentioned and Appendix A for more.
LOADP	h P.FCN LOADP	¬(α, PR)	(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.
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.
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***	PR	(0) Allocates ***n*** local registers ($\leq$ 144) and 16 local flags for the current program. See above.
LocR?	h TEST LocR?	¬α	(-1) Returns the number of local registers allocated.
LOG₁₀	g LG	¬α	(1) Returns the logarithm of x for base 10.
LOG₂	g LB	¬α	(1) Returns the logarithm of x for base 2.
LogF	h MODE LogF	DECM	(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**. With $\xi = \frac{x-\mu}{s}$, Logis returns $F_{Lg}(x) = \frac{1}{1+e^{-\xi}}$ and Logis$_P$ returns $f_{Lg}(x) = \frac{e^{-\xi}}{s \cdot (1+e^{-\xi})^2}$. 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$_P$		
Logis$_u$	h PROB Logis$_u$		
Logis $^{-1}$	h PROB Logis^{-1}		
LOGₓ	g LOGₓ	¬α	(2) Returns the logarithm of y for base x .
	CPX g LOGₓ	DECM	(2) Returns the complex logarithm of $z + i\,t$ for the complex base $x + i\,y$.

Name	Keys to press	in modes	Remarks (see above for general information)
LZOFF	h MODE LZOFF	¬α	(0) Toggles leading zeros like flag 3 does in *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, according to the curve fit model selected. For a straight line, a_0 is the y-intercept and a_1 the slope.
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* etc.	Integer	(-1) Work like MASKL and MASKR on *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 4 returns a mask 11110000_2. You may use it for extracting the four most significant bits from an arbitrary byte by AND.
MASKR			
MAX	h X.FCN MAX	¬α	(2) Returns the maximum of x and y .
MEM?	h TEST 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 becomes 11101000_2 for word size 8).
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 in **X**, a destination row number in **Y**, a source row number in **Z**, and a real number in **T**. It multiples each element m_{zi} 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 in **X**, a row number in **Y**, and a real number in **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 in **X** and two row numbers in **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*.

Name	Keys to press	in modes	Remarks (see above for general information)
MSG	h P.FCN MSG *n*	¬α	(0) Throws the error message specified. See below for the respective error codes. Compare ERR.
M+×	h MATRIX M+×	DECM	(3) Takes two matrix descriptors in **X** and **Y**, and a real number ***z***. Returns $(X)+(Y)\cdot z=(X)$. Thus a scalar multiple of one matrix is added to another matrix. The multiply adds are done in internal high precision and results should be exactly rounded.
M^{-1}	h MATRIX 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 in **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 else.
M-COL	h MATRIX M-COL	DECM	(2) Takes a matrix descriptor in **X** and a column number in **Y**. Returns a loop counter in **X**, dropping the stack. The matrix descriptor is saved in **L**. The loop processes all elements m_{iy} in (X) only. The loop index is DSL if the descriptor is negative and ISG else.
M-DIAG	h MATRIX M-DIAG	DECM	(1) Takes a matrix descriptor in **X**, saves it in **L**, and returns a loop counter in **X**. The loop processes all elements along the matrix diagonal, i.e. all elements m_{ii} in (X). The loop index is DSL if the descriptor is negative and ISG else.
M-ROW	h MATRIX M-ROW	DECM	(2) Takes a matrix descriptor in **X** and a row number in **Y**. Returns a loop counter in **X**, dropping the stack and setting last **L** like all two-argument commands. The loop processes all elements m_{yi} in (X) only. The loop index is DSL if the descriptor is negative and ISG else.
M×	h MATRIX M×	DECM	(3) Takes two matrix descriptors in **Y** and **Z** and the integer part of ***x*** as the base address of the result. Returns $(Z)\cdot(Y)=(X)$. All calculations are done in internal high precision (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 in **Y** and a base register number in **X**. Copies the matrix (Y) into registers starting at **X**. Returns a properly formatted matrix descriptor in **X**.
M.DY	h MODE M.DY	¬α	(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* in **X** and a register number in **Y**. Returns the column that register represents in **Y** and the row in **X**. The descriptor is saved in **L**. M.IJ is similar to *PPC M4*.
M.LU	h MATRIX M.LU	DECM	(1) Takes a *descriptor* of a square matrix in **X**. Transforms (*X*) into its LU decomposition. in-situ. The value in **X** is replaced by a pivot descriptor that defines the pivots that were required to calculate the decomposition. The most significant digit is the pivot for the first diagonal entry, the next most the second and so forth.
M.REG	h MATRIX M.REG	DECM	(3) Takes a matrix *descriptor* in **X**, a row number in **Y**, and a column number in **Z**. The descriptor is saved in **L**. M.REG returns the register number in **X**. It is similar to *PPC M5*.
M.SQR?	h TEST M.SQR?	DECM	(0) Takes a matrix *descriptor* in **X** and tests it. Returns true if the matrix is square.
NAND	h X.FCN NAND	¬α	(2) Works in analogy to AND.
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 *HP-16C*.
nCOL	h MATRIX nCOL	DECM	(1) Takes a matrix *descriptor* in **X**, saves it in **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 [43]. 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	PR	(0) 'Empty' step FWIW.
NOR	h X.FCN NOR	¬α	(1) Works in analogy to AND.

[43] 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** : Norml returns $F_N(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$. See below for Φ.
Norml_P	h PROB Norml_P	DECM	Norml_P [44] returns $f_N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.
Norml_u	h PROB Norml_u	DECM	
Norml^{-1}	h PROB Norml^{-1}	DECM	Norml^{-1} returns x for a given probability F_N in **X**, μ in **J**, and σ in **K**.
NOT	h NOT	Integer	(1) Inverts x bit-wise as on *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 in **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.
OR	h OR	¬α	(2) Works in analogy to AND.
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_n	DECM	(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 $\frac{d}{dx}\left[\left(1-x^2\right)\cdot\frac{d}{dx}f(x)\right] + n(n+1)f(x) = 0$.

[44] 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*) .

Name	Keys to press	in modes	Remarks (see above for general information)
Poiss	h PROB Poiss	DECM	(1) Poisson distribution with the number of successes ***g*** in **X**, the gross error probability ***p₀*** in **J**, and the sample size ***n*** in **K**. The Poisson parameter is $\lambda = n \cdot p_0$ then. See Poisλ below. Poiss^{-1} returns the maximum number of successes ***m*** for a given probability $\boldsymbol{F_P}$ in **X**, $\boldsymbol{p_0}$ in **J**, ***n*** in **K**.
$Poiss_P$	h PROB Poiss$_P$		
$Poiss_u$	h PROB Poiss$_u$		
Poiss $^{-1}$	h PROB Poiss^{-1}		
Poisλ	h PROB Poisλ	DECM	(1) Poisson distribution with ***g*** in **X** (as in Poiss) but with the Poisson parameter ***λ*** in **J**. Poisλ$_P$ [45] 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 $\boldsymbol{F_P}$ in **X** and ***λ*** in **J**.
Poisλ$_P$	h PROB Poisλ$_P$		
Poisλ$_u$	h PROB Poisλ$_u$		
Poisλ $^{-1}$	h PROB Poisλ^{-1}		
PopLR	h P.FCN PopLR	PR	(0) Pops the local registers allocated to the current routine without returning. See LocR and RTN.
PowerF	h MODE PowerF	DECM	(0) Selects the power curve fit model $y = a_0 x^{a_1}$.
PRCL	h P.FCN PRCL	¬α	(0) Copies the current program (from flash or *RAM*) and appends it to *RAM* where it can be edited then (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 $9 \cdot 10^{18}$.
PROFRC	f a b/c	DECM	(1) Sets fraction mode like in *HP-35S*, allowing only proper fractions or mixed numbers in display. Converts x according to the settings by DEN… Absolute decimal equivalents of x must not exceed 100,000. Compare IMPFRC.
		FRC	(1) Allows displaying only proper fractions. Thus converts an improper fraction in **X**, if applicable, e.g. $^5/_3$ into $1\ ^2/_3$.
PROMPT	h P.FCN PROMPT	PR	(0) Displays ***alpha*** and stops program execution (equaling VWα+ X followed by STOP). See above for more.

[45] 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)
PSE	h PSE ***n***	PR	(0) Refreshes the display and pauses program execution for ***n*** ticks, with $0 \le n \le 99$. The pause will terminate early as soon as you press a key.
PSTO	h P.FCN PSTO	¬(α, PR)	(0) Copies the current program from *RAM* and appends it to the flash library. The program must feature at least one LBL statement with an alphanumeric label (preferably at its beginning). If a program with the same label exists in the library already it will be deleted first. Alphanumeric labels present in flash may be browsed by CAT (see *below*) and called by XEQ.
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.
RAD	g RAD	DECM	(0) Sets angular mode to radians.
RAD→	h X.FCN RAD→	DECM	(1) Takes x as radians and converts them to the angular mode currently set.
RAN#	f RAN#	DECM	(-1) Returns a random number between 0 and 1 like RAN in *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 12 −, but without losing a stack level). In analogy, [C]RCL–12 subtracts $r12$ from x and $r13$ from y. See the *addressing table above* for [C]RCL.
RCL–	RCL − ***s***		
RCL×	RCL × ***s***		
RCL/	RCL / ***s***		
RCL↑	RCL ▲ ***s***	¬α	(-1) RCL↑ (↓) recalls the maximum (minimum) of the values in ***s*** and $\mathbf{X}$.
RCL↓	RCL ▼ ***s***		

Name	Keys to press	in modes	Remarks (see above for general information)
RDP	h X.FCN RDP *d*	DECM	(1) Rounds ***x*** to ***d*** decimal places (0 ≤ ***d*** ≤ 99), taking the RM setting into account. See RM.
RDX,	h MODE RDX,	¬α	(0) Sets the decimal mark to a comma.
RDX, / **RDX.**	h ./,	DECM	(0) Toggles the radix mark.
RDX.	h MODE RDX.	¬α	(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*	¬α	(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 TEST REGS?	¬α	(-1) Returns the number of global general purpose registers allocated (0 … 100).
RESET	h P.FCN RESET	¬(α, PR)	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 *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 *HP-16C*, similar to RLn / RLCn there. For RL, 0 ≤ ***n*** ≤ 63 . For RLC, 0 ≤ ***n*** ≤ 64 . See *above* for details of rotating. RL 0 / RLC 0 execute as NOP.
RLC	h X.FCN RLC *n*	Integer	

Name	Keys to press	in modes	Remarks (see *above* for general information)
RM	h MODE RM	¬α	(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 (‘businessman’s rounding’ [46]). 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) Equals RMD on *HP-16C*.
RM?	h TEST 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 *HP-42S*.
		FRC	(1) Rounds x using the current … … denominator like RND in *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 *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	Integer	
RSD	h X.FCN RSD d	DECM	(1) Rounds x to d significant digits, taking the RM setting into account.
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 flash memory, the program pointer will be set to step 000 in *RAM*.

[46] Translates to “kaufmännische Rundung” in German.

Name	Keys to press	in modes	Remarks (see above for general information)
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*** $\in [0;99]$ and ***nn*** = 0 , clearing will stop at the highest allocated global numbered register. For ***sss*** $\in [100;111]$ 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***| there. Destination will be in *RAM* always. **ATTENTION**: For ***sss*** $\in [0;99]$ and ***nn*** = 00 , copying will stop at the highest allocated global numbered register. For ***sss*** $\in [100;111]$ and ***nn*** = 00 , copying will stop at **K**. For ***sss*** ≥ 112 and ***nn*** = 00 , copying will stop at the highest allocated local register.
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**: For ***sss*** $\in [0;99]$ and ***nn*** = 0 , sorting will stop at the highest allocated global numbered register. For ***sss*** $\in [100;111]$ and ***nn*** = 0 , sorting will stop at **K**. For ***sss*** ≥ 112 and ***nn*** = 0 , sorting will stop at the highest allocated local register.
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.

Name	Keys to press	in modes	Remarks (see *above* for general information)
R↑	h R↑	¬α	Rotates the stack contents one level up or down, respectively. See *above* for details.
R↓	R↓		
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.
SAVE	h P.FCN SAVE	¬(α, PR)	(0) Saves user program space, registers and system state to flash memory, returns Saved then. 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 by n decimal positions, equivalent to multiplying x by 10^n .
SDR	h X.FCN SDR ***n***	DECM	(1) Shifts digits right by n decimal positions, equivalent to dividing x through 10^n .
SEED	h STAT SEED	DECM	(0) Stores a seed for random number generation.
SENDA	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.
SENDP			
SENDR			
SENDΣ			
SEPOFF	h MODE SEPOFF	¬α	(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.
SEPOFF / SEPON	h ./,	Integer	
SEPON	h MODE SEPON	¬α	
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 $\bar{y}$).

Name	Keys to press	in modes	Remarks (see above for general information)
$SERR_w$	h STAT SERRw	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	¬α	(0) Sets some regional preferences (see above).
SETDAT	h MODE SETDAT	DECM	(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.	¬α	(0) Set some regional preferences (see above).
SETTIM	h MODE SETTIM	DECM	(0) Sets the time for the real time clock (the emulator takes this information from the PC clock).
SETUK SETUSA	h MODE SETUK etc.	¬α	(0) Set some regional preferences (see above).
SF	f SF *n*	¬α	(0) Sets the flag specified.
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.
	CPX h X.FCN cSIGN	DECM	(1) Returns the unit vector of $x + i\,y$ in **X** and **Y**.
SIGNMT	h MODE SIGNMT	¬α	(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.
SL	h X.FCN SL *n*	Integer	(1) Works like *n* (≤ 63) consecutive SLs on *HP-16C*. See above for details of shifting. 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 *HP-15C*. This also means SLV acts as a test, so the next program step will be skipped if SLV failed to find a root. Please refer to the *HP-15C Owner's Handbook* (Section 13 and Appendix D) for more information about automatic root finding.
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 TEST 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 *HP-16C*. See above for details of shifting. SR 0 executes as NOP.
sRCL	h X.FCN sRCL ***s***	¬α	(-1) Assumes the source ***s*** containing single precision data and recalls them as such. See below.
SSIZE4	h MODE SSIZE4	¬α	Set the stack size to 4 or 8 levels, respectively. See above. Please note register contents will remain unchanged in this operation. The same will happen if stack size is modified by any other operation.
SSIZE8	h MODE SSIZE8		
SSIZE?	h TEST SSIZE?	¬α	(-1) Returns the number of stack levels allocated.

Name	Keys to press	in modes	Remarks (see *above* for general information)
STO	STO ***d***	¬α	(0) See the *addressing table above* for ᶜSTO.
STOM	STO MODE ***s***	¬α	(0) Stores mode settings for later use as described *above*. Take RCLM to recall them.
STOP	R/S	PR	(0) Stops program execution. May be used to wait for input, for example.
STOPW	see *below*	DECM	Stopwatch application following HP-55, see *below*.
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 keystrokes RCL 12 x⇄y − STO 12 would do, but without touching the stack at all. See the *addressing table above* for ᶜSTO.
STO–	STO − ***d***	¬α	
STO×	STO × ***d***	¬α	
STO/	STO / ***d***	¬α	
STO↑	STO ▲ ***d***	¬α	(0) STO↑ (↓) takes the maximum (minimum) of the values in ***d*** and **X** and stores it.
STO↓	STO ▼ ***d***	¬α	
SUM	h STAT SUM	DECM	(-2) Recalls the linear sums ***Σy*** and ***Σx***. Useful for elementary vector algebra in 2D.
s_W	h STAT s_w	DECM	(-1) Calculates the standard deviation for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+): $s_w = +\sqrt{\frac{\sum y_i \cdot \sum(y_i \cdot x_i^2) - [\sum(y_i \cdot x_i)]^2}{(\sum y_i)^2 - \sum y_i^2}}$.
s_{XY}	h STAT s_{xy}	DECM	(-1) Calculates the sample covariance for the two data sets entered via Σ+. It depends on the fit model selected. For LinF, it returns $s_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n \cdot (n-1)}$. See COV for the population covariance.

Name	Keys to press	in modes	Remarks (see above for general information)
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***.
TICKS	h P.FCN TICKS	¬α	(-1) Returns the number of ticks from the real time clock at execution time. With the quartz built in, 1 tick = 0.1 s. Without, it may be 10% more or less. So the quartz is an inevitable prerequisite for the clock being useful in medium to long range.
TIME	h X.FCN 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_n	DECM	(2) Chebychev's (a. k. a. Čebyšev, Tschebyschow, Tschebyscheff) polynomials of first kind $T_n(x)$ with ***n*** in **Y**, solving the differential equation $\left(1-x^2\right)y''-x\cdot y'+n^2y=0$.
TOP?	h TEST TOP?	PR	(0) Executes the next step only if TOP? is called in a program that isn't a subroutine, i.e. if the program-running flag is set and the subroutine return stack pointer is clear.
TRANSP	h MATRIX TRANSP	DECM	(1) Takes a matrix descriptor in **X** and returns the descriptor of its transpose. The transpose is done in-situ and does not require any additional memory.
$t_p(x)$	h PROB $t_p(x)$	DECM	(1) Student's t distribution. $t_u(x)$ equals Q(t) and $t^{-1}(p)$ equals t p in *HP-21S*. The degrees of freedom are stored in **J**. See above for an application.
$t_u(x)$	h PROB $t_u(x)$		
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 the machine in the mode set. Thus, in integer mode, 1 is returned.
U_n	h X.FCN U_n	DECM	(2) Chebychev's polynomials of second kind $U_n(x)$ with ***n*** in **Y**, solving the differential equation $\left(1-x^2\right)y''-3x\cdot y'+n(n+2)y=0$.
UNSIGN	h MODE UNSIGN	¬α	(0) Sets unsigned mode like UNSGN on *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 the next key is pressed, working similar to AVIEW in *HP-42S*. See above for more.
	h VIEW −	α	
VWα+	h VIEW s	α	(0) Displays ***alpha*** in the top row plus 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 and returns the name of the day in the dot matrix and a corresponding integer in the numeric display (Monday = 1, Sunday = 7) [47].
W_m	h X.FCN Wm	DECM	(1) W_p returns the principal branch of Lambert's W for given $x \geq -1/e$. W_m returns the negative branch.
W_p	h X.FCN Wp		
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 the shape parameter ***b*** in **J** and the characteristic lifetime ***T*** in **K**:
Weibl_P	h PROB WeiblP		Weibl_P [48] returns $f_W(t) = \frac{b}{T}\left(\frac{t}{T}\right)^{b-1} e^{-\left(\frac{t}{T}\right)^b}$,
Weibl_u	h PROB Weiblu		Weibl returns $F_W(t) = 1 - e^{-\left(\frac{t}{T}\right)^b}$.
Weibl^{-1}	h PROB Weibl-1		Weibl^{-1} returns the survival time t_s for given probability F_W, ***b*** in **J** and ***T*** in **K**.
WSIZE	h MODE WSIZE *n*	¬α	(0) Works like on *HP-16C*, but with the parameter following the command instead of taken from **X**. Reducing the word size truncates the values in the stack registers employed, including **L**. WSIZE 0 sets the word size to maximum, i.e. 64 bits.
WSIZE?	h TEST WSIZE?	¬α	(-1) Recalls the word size set.

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

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

Name	Keys to press	in modes	Remarks (see above for general information)
x^2	g x^2	¬α	(1)
x^3	h X.FCN x³	¬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 less if there are less) as a label and calls or executes the respective routine.
XNOR	h X.FCN XNOR	¬α	(2) Works in analogy to AND.
XOR	h XOR	¬α	(2) Works in analogy to AND.
XTAL ?	h TEST XTAL?	¬α	(0) Tests for presence of the crystal necessary for a precise real time clock (think of Xmas).
x̄	f x̄	DECM	(-2) Pushes $\bar{y} = \frac{1}{n}\sum y$ and $\bar{x} = \frac{1}{n}\sum x$ on the stack. See also s, SERR, and σ.
x̄g	h STAT x̄g	DECM	(-2) Puts $\bar{y}_g = \sqrt[n]{\prod y} = e^{\frac{1}{n}\sum \ln y}$ and $\bar{x}_g = \sqrt[n]{\prod x}$ on the stack, i.e. the geometric means. See also ε, ε_m, and ε_P.
x̄w	h STAT x̄w	DECM	(-1) Returns the arithmetic mean $\bar{x}_w = \sum xy \big/ \sum y$ for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+). See also s_w and $SERR_w$.
x̂	h STAT x̂	DECM	(1) Returns a forecast ***x*** for a given ***y*** (in **X**) following the fit model chosen. See L.R. for more.
$\sqrt[x]{y}$	h X.FCN $\sqrt[x]{y}$	¬α	(2)
x !	h !	DECM	(1) Returns $\Gamma(x+1)$.
		Integer	(1) Returns the factorial $n!$.

Name	Keys to press	in modes	Remarks (see above for general information)
x → α	h X.FCN x→α	All	(0) Interprets ***x*** as character code. Appends the respective character to ***alpha***, similar to XTOA in *HP-42S*.
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, and the like.
x⇆ Y	x⇆y	¬α	Swaps ***x*** and ***y***, performing Re⇆Im if a complex operation was executed immediately before. See above for ᶜ***x⇆y*** .
x < ?	h TEST x<_? ***a***	¬α	(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 in the addressing table 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. Then, 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 in the addressing table above.
x ≤ ?	h TEST x≤_? ***a***	¬α	
x = ?	f x = ? ***a***	¬α	
x = +0 ?	h TEST x=+0?	¬α	
x = –0 ?	h TEST x=–0?	¬α	
x ≈ ?	h TEST x≈_? ***a***	¬α	
x ≠ ?	g x ≠ ? ***a***	¬α	
x ≥ ?	h TEST x≥_? ***a***	¬α	
x > ?	h TEST x>_? ***a***	¬α	
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.
y^x	C	¬(α, 13, 14, 15, h)	(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	¬α	(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)
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 less if there is only less) 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 *HP-42S*.
αLENG	h X.FCN αLENG	All	(-1) Returns the number of characters found in ***alpha***, like ALENG in *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***.
αOFF	h P.FCN αOFF	PR	(0) Work like AOFF and AON in *HP-42S*, turning alpha mode off and on.
αON	h P.FCN αON		
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***. If e.g. 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 *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 *HP-42S*. αSL 0 equals NOP.
αSR	h X.FCN αSR ***n***	All	(0) Works like αSL but takes the ***n*** rightmost characters instead.

Name	Keys to press	in modes	Remarks (see *above* for general information)
αSTO	f STO *d*	α	(0) Stores the first (i.e. leftmost) 6 characters of ***alpha*** in destination ***d*** .
	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 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 less if there are only less) of the converted code as an alpha label and calls or executes the respective routine.
α → x	h X.FCN α→x	All	(-1) Returns the character code of the leftmost character in ***alpha*** and deletes this character, like ATOX in *HP-42S*.
β	h X.FCN β	DECM	(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 for avoiding ambiguities.
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 incomplete gamma function $\gamma(x,y)=\int_0^y t^{x-1}e^{-t}dt$, or $\Gamma_u(x,y)=\int_y^\infty t^{x-1}e^{-t}dt$ for the upper incompl. gamma function, respectively.
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 *HP-12C*.
Δ%	g Δ%	DECM	(2) Returns $100\cdot\frac{x-y}{y}$ like %CH in *HP-42S*.
ε	h STAT ε	DECM	(-2) Returns the scattering factors (a.k.a. geometric standard deviations) for log-normally distributed data $\ln(\varepsilon_y)=\sqrt{\frac{\sum \ln^2(y)-2n\cdot\ln(\bar{y}_G)}{n-1}}$ and $\ln(\varepsilon_x)$. This ε works for the geometric mean $\bar{x}_g$ in analogy to the standard deviation s for the arithmetic mean x̄ but multiplicative instead of additive.

Name	Keys to press	in modes	Remarks (see *above* for general information)
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 for real arguments, with $\zeta(x)=\sum_{n=1}^{\infty}\frac{1}{n^x}$ for $x>1$, and its analytical continuation for $x<1$: $$\zeta(x)=2^x\pi^{x-1}\sin\left(\frac{\pi}{2}x\right)\cdot\Gamma(1-x)\cdot\zeta(1-x).$$
π	h π	DECM	(-1) Recalls π.
	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 `cccccc.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 `cccccc` by `ii` and runs said routine again if then `cccccc` ≥ `fff` , else returns the resulting product in **X**.
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 in **X**.

Name	Keys to press	in modes	Remarks (see above for general information)
$\Sigma \ln^2 x$, $\Sigma \ln^2 y$, $\Sigma \ln x$, $\Sigma \ln xy$, $\Sigma \ln y$, $\Sigma x \ln y$, $\Sigma y \ln x$	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 enhances readability of programs significantly. Please 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 not numeric.
σ_w	h STAT σw	DECM	(-1) Works like s_w but returns the standard deviation of the population instead. $\sigma_w = +\sqrt{\frac{\sum y_i (x_i - \bar{x}_w)^2}{\sum y_i}}$
Σx, 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 enhances readability of programs significantly. Please note these sums are stored in special registers in your *WP 34S*.
Σ+	h Σ+	DECM	Adds a data point to the statistical sums. — Both functions disable stack lift as usual. Both may be used for 2D vector algebra as well.
	A	DECM	Shortcut working if label A is not defined.
Σ–	h Σ–	DECM	Subtracts a data point from the statistical sums.
$\Phi_u(x)$	h PROB Φu(x)	DECM	(1) Standard normal error probability $\Phi_u(x) = \int_x^\infty \varphi(\tau) d\tau$, equals Q in *HP-32E* and Q(z) in *HP-21S*.
φ(x)	h PROB φ(x)	DECM	(1) Standard normal pdf $\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.

Name	Keys to press	in modes	Remarks (see above for general information)
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 *HP-32E* and z_p in *HP-21S*.
χ^2	h PROB χ^2	DECM	(1) Chisquare 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 *HP-21S*.
χ^2INV	h PROB χ^2INV		
χ^2_P	h PROB χ^2_P		
χ^2_u	h PROB χ^2_u		
(-1)^x	h X.FCN $(-1)^x$	¬α	(1) For x not being a natural number, this function will return $\cos(\pi\cdot x)$.
+	+	¬α	(2) Returns $y+x$.
	CPX +	DECM	(2) Returns $[\,x+z,\ \ y+t,\ \ \ldots\,]$. May be employed for adding 2D vectors as well.
–	–	¬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.
/	/	¬α	(2) Returns $y\,/\,x$. If the division remainder is ≠ 0 in integer modes, carry will be set.
	CPX /	DECM	(2) Returns $[\,\frac{x\cdot z+y\cdot t}{z^2+t^2},\ \frac{z\cdot y-x\cdot t}{z^2+t^2},\ \ldots\,]$.
+/–	+/–	¬α	(1) 'Unary minus', corresponding to $x\cdot(-1)$ or $x_c\cdot(-1)$, respectively.
→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.

Name	Keys to press	in modes	Remarks (see *above* for general information)
→GRAD	→ GRAD	DECM	(1) Like →DEG, but converts to gon or grads.
→H	→ f H.d	DECM	(1) Takes x as hours or degrees in the format `hhhh.mmssdd` and converts them into a decimal time or angle.
→H.MS	→ f H.MS	DECM	(1) Takes x as decimal hours or degrees and converts them into 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 $(\boldsymbol{x}, \boldsymbol{y})$ of a point or a vector and converts them to the respective polar coordinates / components $(\boldsymbol{r}, \boldsymbol{\vartheta})$ 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 $(\boldsymbol{r}, \boldsymbol{\vartheta})$ of a point or such components of a vector and converts them to the respective Cartesian coordinates or components $(\boldsymbol{x}, \boldsymbol{y})$.
%	f %	DECM	(1) Returns $\frac{x \cdot y}{100}$, leaving **Y** unchanged.
%MG	h X.FCN %MG	DECM	(2) Returns the margin [49] $100 \cdot \frac{x-y}{x}$ in % for a price x and cost y , like %MU-Price in *HP-17B*.
%MRR	h X.FCN %MRR	DECM	(3) Returns the mean rate of return in percent per period, i.e. $100 \cdot \left[\left({}^{x}\!/_{y} \right)^{1/z} - 1 \right]$ 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 $100 \cdot \frac{x}{\sum x}$.

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

Name	Keys to press	in modes	Remarks (see above for general information)
%+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 *HP-17B*. Formula: $\frac{y}{1-\frac{x}{100}}$
√‾	f √x	¬α	(1) If the input is no square number, carry will be set in integer modes.
	D	¬(α, 14, 15, h)	(1) Shortcut working if label D is not defined.
∫	g ∫ *label*	DECM	Integrates the function given in the routine specified. Lower and upper integration limits must be supplied in **Y** and **X**, respectively. Otherwise, the user interface is as in *HP-15C*. 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.
||	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	All	(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. The printer will actually print only when a line feed is sent to it.
🖶CHAR	h P.FCN 🖶CHAR ***n***	All	(0) Sends a single character with the code specified to the printer. Character codes $n > 127$ can only be specified indirectly. See 🖶ADV.
🖶$^{c}r_{XY}$	h P.FCN 🖶ᶜrXY ***r***	All	(0) Prints the register specified and the next one, i.e. prints an entire complex number. A comma will separate both components. Works like 🖶r else. **Example**: Assume 🖶MODE 1, SCI 1, $x = -1.2$ and $y = 0.34$. Then the output of 🖶ᶜrXY 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.

Name	Keys to press	in modes	Remarks (see above for general information)
🖶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 instead, resulting in some jitter on the printout but packing more characters in a single line. 2: Use the small display font which allows for packing even more info on a single 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	All	(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*	All	(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	All	(1) Interprets x in the form sss.nn. Prints the contents ***nn*** registers starting with number ***sss***. Each register takes one line starting with a label. **ATTENTION**: For $sss \in [0;99]$ and $nn = 0$, printing will stop at the highest allocated global numbered register. For $sss \in [100;111]$ and $nn = 0$, printing will stop at **K**. For $sss \geq 112$ and $nn = 0$, printing will stop at the highest allocated local register. See also 🖶ADV.
🖶STK	h P.FCN 🖶STK	All	(0) Prints the stack contents. Each level prints in one line starting with a label. See 🖶ADV.
🖶TAB	h P.FCN 🖶TAB *n*	All	(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	All	(-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 respective font.

Name	Keys to press	in modes	Remarks (see *above* for general information)
🖶α	h P.FCN 🖶α	All	(0) Appends ***alpha*** to the print line, trailed by a line feed.. See 🖶ADV.
🖶α+	h P.FCN 🖶α+	All	(0) Sends ***alpha*** to the printer without a trailing line feed, allowing to append further information to this line. May be repeated. See also 🖶ADV, 🖶r and 🖶+α.
🖶Σ	h P.FCN 🖶Σ	All	(0) Prints the summation registers. Each register prints in one line starting with a label. See 🖶ADV.
🖶+α	h P.FCN 🖶+α	All	(0) Appends ***alpha*** to the print line, adjusted to the right and trailed by a line feed. See 🖶ADV. **Example:** The following program section CLα α'Lef' α t 🖶α+ CLα α'Rig' α'ht' 🖶+α will print, if 🖶MODE 1 is set: Left Right
🖶?	h TEST 🖶?	¬α	(0) Tests if the necessary hardware and software is installed for printing.
🖶#	h P.FCN 🖶# *n*	All	(0) Sends a single byte without any translation to the printer (e.g. a control code). ***n*** > 127 can only be specified indirectly. See 🖶ADV.
#	h CONST # ***n***	PR	Inserts an integer 0 ≤ ***n*** ≤ 255 in a single step, thus saving up to two steps and an ENTER.
	CPX h CONST # ***n***	DECM	Works like in real domain but clears ***y*** in addition. The shortcut works for 1 ≤ ***n*** ≤ 9 only.
	CPX ***n***		

Non-programmable Control, Clearing and Information Commands

Keys to press	in modes	Remarks
f 1/x	In CONV	Inverts the current conversion (see below).
8 [50]	Asking for confirmation	Denies the question Sure? with N for 'no'. Any other input except R/S = Y will be ignored.
	α	Appends an '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	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.
h OFF	¬PR	Turns your *WP 34S* off.
	PR	Enters the command OFF as described above.
ON	Calculator off	Turns your *WP 34S* on.
	Else	ON-key combinations are found in the appendices. Of them, most important are ON + + or − adjusting display contrast.
h P/R	¬α	Toggles programming mode (PR).

[50] 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. Period.

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

Keys to press	in modes	Remarks
R/S	Asking for confirmation	Confirms the question Sure? with Y for 'yes'. Any other input except 8 = N will be ignored.
	CAT open, ¬PR	Runs the program whose label is displayed (compare XEQ below 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 an '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 ... … the label displayed (see *below*).
	Else	Acts like the command XEQ described above.
f α	¬PR ¬α	Enters alpha mode for appending characters to ***alpha***. For starting a new string, use CLα first.
	¬PR α	Leaves alpha mode.
	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 e.g. α?) 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 [51].

[51] Note alpha group mode is left automatically after 3 characters put in, so it must be called again for continuation.

Keys to press	in modes	Remarks
f ↑	Catalog open	Calls the character ⎙.
	α	Toggles upper and lower case (the latter is indicated by the annunciator ⬇).
	Else	Calls ⎙r X (see there).
←	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.
▲ / ▼	Status display open	Goes to previous / next status window. See above.
	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 less 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 *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 respective program step, but the keys will not repeat.
f ◀ / g ▶	Integer & ¬PR	Shifts the display window to the left / right like in *HP-16C*. Helpful while working with small bases. See above.
	DECM & ¬PR	Shows the full mantissa until the next key is pressed. See above. Compare the command SHOW in previous calculators.
→ f 2	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.
→ 8		
→ 16		

Alphanumeric Input

Character	Keys to press	in modes	Remarks
␣	h PSE	α	Appends a blank space to ***alpha***.
°	.	DECM	Separates degrees or hours from minutes and seconds, so input format is `hhhh.mmssdd`. The user has to take care where an arbitrary real number represents such an angle or time.
0 … 9	0 … 9	¬α	Standard numeric input. For integer bases <10, input of illegal digits is blocked. Please note you cannot enter more than 12 digits in the mantissa.
		in addressing	Register input. See the *tables* above for the number ranges.
	0, 1, f 2, … , f 9	α	Appends the respective digit to ***alpha***.
A … F	A … F (grey print)	11, 12, 13, 14, 15, h	Numeric input for digits >10. See *above* for more information.
A … Z	A … Z (grey print)	in addressing	Register input. See the *virtual keyboard* above for the letters applicable.
		α	Appends the respective Latin letter to ***alpha***. Use f ↑ to toggle cases.
E	EEX	DECM & ¬FRC	Works like E in the Pioneers.
i	CPX .	DECM & ¬FRC	Enters complex number ***i*** , i.e. $x = 0$ and $y = 1$.
Α … Ω	g A … g O (grey print)	α	Appends the respective Greek letter to ***alpha***. Use f ↑ to toggle cases. See *above* for more.
(	f ◄(	α	Appends the respective symbol to ***alpha***.
)	g)►		
+	f +		
–	f –		
x	f ×		

Character	Keys to press	in modes	Remarks
/	Second .	DECM	A persistent 2nd . in input turns to fraction mode and will be interpreted as explained below. Please note you cannot enter EEX after you entered . twice – but you may delete the 2nd dot while editing the input line.
		FRC	First . is interpreted as a space, 2nd as a fraction mark. 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	α	Appends the respective symbol to ***alpha***.
.	f .	α	Appends the respective symbol to ***alpha***.
'.' or ','	.	DECM	Inserts a radix mark as selected.
!	h !	α	Appends the respective symbol to ***alpha***.
?	h ▼	α	Appends the respective symbol to ***alpha***.
⇆	h x⪌	α	Appends the respective symbol to ***alpha***.
≠	h XOR	α	Appends the respective symbol to ***alpha***.
&	h AND	α	Appends the respective symbol to ***alpha***.
\	h /	α	Appends the respective symbol to ***alpha***.
|	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 for EXITing this first.

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

Additionally, there are some ***browsers*** allowing to check memory, flags, program labels and registers (i.e. CAT, SHOW, STATUS, and SUMS). ▲, ▼, EXIT, and ← work in all browsers 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, there is one **application** provided:

Name	Keys to press	in modes	Remarks
STOPW	h X.FCN STOPW CPX R/S	¬PR	Stopwatch following the *HP-55* timer. This works only with a quartz and proper firmware installed (or on the emulator). {The content of **X** will be taken as start time.} [52] Starting STOPW, the display will look like this:

unless the timer was running before already. Within STOPW:

R/S will start or stop the timer without changing its value.

CLx or ⬅ will reset the timer to zero without changing its status.

EEX will hide or display tenths {hundredths} of seconds. Startup default is 'display'.

n n will set the '*current register address*' (*CRA*, startup default is 00 [53]). Already your numeric input will be displayed in the exponent section like shown here [54]:

ENTER↑ will store the present timer value in the current register at execution time in format `hhh.mmssd` without changing the timer status or value. It will then increment the *CRA* and display it like shown above.

→ will hide or display the *CRA*. Startup default is 'hide'.

▲ and ▼ will increment or decrement the *CRA*, respectively.

. combines ENTER↑ and CLx / ⬅, i.e. it will store the time counted at *CRA* and reset the timer to start a new count.

These keystrokes are not featured in *HP-55.*

RCL ***nn*** will recall ***rnn*** without changing the status of the timer. The value recalled may be used e.g. as start time for further incrementing.

EXIT will leave the application. The timer will continue 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.

Please note there are <u>no other keys</u> working in STOPW – so e.g. for adding or subtracting split times you have to leave the application.

---

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

[53] On the HP-55, input of a single digit was sufficient for storing, since only 10 registers were featured for this purpose. Furthermore, there was no automatic address increment.

[54] Attempts to specify a *CRA* out of allocated address range will be blocked and may cause '_ _' or alike being displayed in the exponent section.

The catalogs and browsers your *WP 34S* features are listed below:

Keys to press	in modes	Contents and special remarks
h CAT	¬α	Defined alphanumeric labels. The first label shown is the top global label of the current program – if there is none, its end is shown. ▲ and ▼ browse global labels, while the location of the respective label is indicated in the lower line (rAM, Lib for the library in flash memory, or buP for the backup region). Duplicate labels will show the primary address in addition, 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 routine (with routines 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.
h CONST	DECM & ¬PR	Constants like in *HP-35s*, but more. See them listed *below*. Picking a constant will recall it.
	Integer	Calls the command # (see *above*).
	DECM & PR	Picking a constant will insert a program step beginning with # followed by the name of the constant selected. It will then recall it 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.
h CONV	DECM	Conversions as listed in a *table below*.
f CPX	α	'Complex' letters mandatory for many languages (see *below*). Case may be toggled here (see f ↑ *above*).
h MATRIX	DECM	Matrix operations library.
h MODE	¬α	Mode setting functions.
h PROB	DECM	Probability distributions beyond standard normal and its inverse.
h P.FCN	All	Extra programming and I/O functions.
h R↑	α	Superscripts and subscripts (see *below*).

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 jumps to the corresponding register, as does any legal two-digit number (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 – please see above.
h STAT	DECM	Extra statistical functions.
h STATUS	¬α	Shows the memory status and browses the status of all user flags, similar to STATUS on *HP-16C*. See above for a detailed description.
h SUMS	DECM	All summation registers and their 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. These three catalogs are merged in mode PR to ease programming.
	Integer	Extra integer functions.
	α	Extra alpha functions.
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. Within each catalog, items are sorted alphabetically (see above for the sorting order). You may access particular items fast and easily by typing the first characters of their names – see below for some examples and constraints.

Catalog Contents in Detail

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

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

CFALL
CLALL
CLPALL
CLREGS
CLSTK
CLα
DEC
DROP
DSL
DSZ
END
ERR
FF
f ’(x)
f ”(x)
GTOα
INC
ISE
ISZ
LOAD
LOADP
LOADR
LOADSS
LOADΣ
LocR
MSG
NOP
PopLR
PRCL
PROMPT
PSTO
PUTK
RCLS
RECV
RESET
R-CLR
R-COPY
R-SORT
R-SWAP
SAVE
SENDA
SENDP
SENDR
SENDΣ
STOS
TICKS
t⇆
VIEWα
VWα+
XEQα
y⇆
z⇆
αGTO
αOFF
αON
αXEQ
⎙ADV
⎙CHAR
$⎙^{C}r_{XY}$
⎙PROG
⎙r
⎙REGS
⎙STK
⎙TAB
⎙WIDTH
⎙α
⎙α+
⎙Σ
⎙+α
⎙#

STAT

COV
L.R.
SEED
SERR
SERR_W
SUM
s_W
s_{XY}
x̄g
$\bar{x}_W$
x̂
ε
ε_m
ε_P
σ
σ_W
%Σ

SUMS

nΣ
$\Sigma\ln^2x$
$\Sigma\ln^2y$
Σlnx
Σlnxy
Σlny
Σx
Σx^2
Σx^2y
Σxlny
Σxy
Σy
Σy^2
Σylnx

TEST

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

X.FCN varies with the mode set, except in programming [55]. It contains in …

… alpha mode:

VERS
x→α
αDATE
αDAY
αIP
αLENG
αMONTH
αRC#
αRL
αRR
αSL
αSR
αTIME
α→x

… decimal mode:

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

… integer modes:

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

CPX **X.FCN**

$^C\sqrt[3]{x}$
^{C}AGM
^{C}CONJ
^{C}CROSS
^{C}DOT
^{C}DROP
$^Ce^x$ -1
^{C}FIB
^Cg_d
$^Cg_d^{-1}$
^{C}LN1+x
^{C}LNβ
^{C}LNΓ
^{C}SIGN
^{C}SINC
^CW_p
$^CW^{-1}$
$^Cx^3$
$^C\sqrt[x]{y}$
$^C\beta$
$^C\Gamma$
$^C(-1)^x$

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

À	À	À	à	à	à
Á	Á	Á	á	á	á
ÂÃĀĂ	ÂÃĀĂ	ÂÃĀĂ	âãāă	âãāă	âãāă
Ä	Ä	Ä	ä (ă)	ä	ä
Æ	Æ	Æ	æ	æ	æ
Å	Å	Å	å	å	å
Ć	Ć	Ć	ć	ć	ć
Č	Č	Č	č	č	č
Ç	Ç	Ç	ç	ç	ç
Đ	Đ	Đ	đ	đ	đ
È	È	È	è	è	è
É	É	É	é	é	é
ÊĒĚĔ	ÊĒĚĔ	ÊĒĚĔ	êēěĕ	êēěĕ	êēěĕ
Ë	Ë	Ë	ë (ĕ)	ë	ë
	Ħ	Ħ	ħ	ħ	ħ
Ì	Ì	Ì	ì	ì	ì
Í	Í	Í	í	í	í
ÎĨĪĬ	ÎĨĪĬ	ÎĨĪĬ	îĩīĭ	îĩīĭ	îĩīĭ
Ï	Ï	Ï	ï (ĭ)	ï	ï
ÑŇ	ÑŇ	ÑŇ	ñň	ñň	ñň
Ò	Ò	Ò	ò	ò	ò
Ó	Ó	Ó	ó	ó	ó
ÔÕŌŎ	ÔÕŌŎ	ÔÕŌŎ	ôõōŏ	ôõōŏ	ôõōŏ
Ö	Ö	Ö	ö (ŏ)	ö	ö
Ø	Ø	Ø	ø	ø	ø
Ř	Ř	Ř	ř	ř	ř
Š	Š	Š	š	š	š
	ß	ß	ß	ß	ß
Ù	Ù	Ù	ù	ù	ù
Ú	Ú	Ú	ú	ú	ú
ÛŨŪŬ	ÛŨŪŬ	ÛŨŪŬ	ûũūŭ	ûũūŭ	ûũūŭ
Ü	Ü	Ü	ü (ŭ)	ü	ü
Ů	Ů	Ů	ů	ů	ů
	X̂ / X̄	X̂ / X̄	x̂ / x̄	x̂ / x̄	x̂ / x̄
Ý	Ý	Ý	ý	ý	ý
	Ŷ / Ȳ	Ŷ / Ȳ	ŷ / ȳ	ŷ / ȳ	ŷ / ȳ
Ÿ	Ÿ	Ÿ	ÿ	ÿ	ÿ
Ž	Ž	Ž	ž	ž	ž

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

./.																		
,	:	;	'	"	*	@	_	~	%	$	€	£	¥	⊙	⊕	⎙	`	#
,	:	;	'	"	*	@	_	~	%	$	€	£	¥	⊙	⊕	⎙	`	#
,	:	;	'	"	*	@	_	~	%	$	€	£	¥	⊙	⊕	⎙	`	#

TEST									
<	≤	=	≈	≥	>	[	]	{	}
<	≤	=	≈	≥	>	[	]	{	}
<	≤	=	≈	≥	>	[	]	{	}

→								
→	←	↑	↓	√	∫	∞	^	↕
→	←	↑	↓	√	∫	∞	^	↕
→	←	↑	↓	√	∫	∞	^	↕

R↑																								
0	°	1	2	^{2}	^{3}	A	B	c	^{c}	d	e	k	m	n	p	q	u	w	x	^{x}	γ	μ	$^{-1}$	*	∞
0	°	1	2	^{2}	^{3}	A	B	c	^{c}	d	e	k	m	n	p	q	u	w	x	^{x}	γ	μ	$^{-1}$	*	∞
0	°	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 $3\cdot10^9$ 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, Slovenčina, Slovenščina, Srpski, Basa Sunda, Suomeksi, Svenska, Tagalog, Winaray, and Zhōngwén (with a little trick explained below). If you know further living languages covered, please tell us.

Mandarin Chinese (Zhōngwén) features four tones, usually transcribed like e.g. mā, má, 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.

Find the full character set provided in Appendix D.

Accessing Catalog Items the Fast Way

Each and every catalog may be browsed by 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. à 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. ú)	
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 reaching the item desired (e.g. FS?C).	(e.g. ü).	(e.g. €).
n	User input	XEQ or ENTER↑		
		Your *WP 34S* leaves the catalog returning to the mode set before … … and executes or inserts the command chosen, or recalls the constant selected.	… and appends the selected character to ***alpha***.	
	Dot matrix display	**Result** (e.g. true)	**Contents of alpha register** (e.g. 3 Rüben à 0,25€)	

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

Constants (CONST)

Your *WP 34S* contains a rich set of constants. Navigation therein works as explained above. Names of astronomical and mathematical constants are printed on colored background below. Values of physical constants (*incl. their relative standard deviations given in parentheses below*) are from CODATA 2010, copied in July 2011, unless stated otherwise explicitly. Green background denotes exact or almost exact values. The more the color turns to red, the less precise the respective constant is known, even by the national standards institutes and the international scientific community [56].

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	a	365.242 5 d *(per definition)*	Gregorian year
a0	a0	5.291 772 109 2**E**-11 *(3.2**E**-10) m*	Bohr radius $a_0 = \frac{\alpha}{4\pi \cdot R_\infty}$
am	am	384.4**E**6 *(1**E**-3) m*	Semi-major axis of the Moon's orbit
a⊕	a⊕	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	c	2.997 924 58**E**8 m/s *(per definition)*	Vacuum speed of light ≈ 300 000 km/s
c1	c1	3.741 771 53**E**-16 *(4.4**E**-8)* $m^2 \cdot W$	First radiation constant $c_1 = 2\pi \cdot h \cdot c^2$
c2	c2	0.014 387 770 *(9.1**E**-7)* $m \cdot K$	Second radiation constant $c_2 = hc/k$
e	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	eE	2.718 281 828 459 045…	Euler's e. Please note the letter e represents the electron charge elsewhere in this table.

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

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

		Numeric value	Remarks
$m_u c^2$	$m_u c^2$	1.492 417 954**E**-10 *(4.4**E**-8)* J	Atomic unit mass energy equivalent
m_μ	m_μ	1.883 531 475**E**-28 *(5.1**E**-8)* kg	Muon mass
$M_\odot$	$M_\odot$	1.989 1**E**30 *(5**E**-5)* kg	Mass of the sun
$M_\oplus$	$M_\oplus$	5.973 6**E**24 *(5**E**-5)* kg	Mass of the Earth
N_A	N_A	6.022 141 29**E**23 *(4.4**E**-8)* $/ \, mol$	Avogadro's number
NaN	NaN	not nuMEric	'Not a Number', e.g. $\ln(x)$ for $x \leq 0$.
p_0	p_0	101 325 Pa *(per definition)*	Standard atmospheric pressure
q_p	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	R	8.314 462 1 *(9.1**E**-7)* $\frac{J}{mol \cdot K}$	Molar gas constant
r_e	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	R_K	25 812.807 443 4 *(3.2**E**-10)* Ω	von Klitzing constant $R_K = h / e^2$
R_m	R_m	1.737 530**E**6 *(5**E**-7)* m	Mean radius of the Moon
R_∞	R_∞	1.097 373 156 853 9**E**7 *(5.0**E**-12)* $/ \, m$	Rydberg constant $R_\infty = \alpha^2 m_e c / 2h$
$R_\odot$	$R_\odot$	6.96**E**8 *(5**E**-3)* m	Mean radius of the sun
$R_\oplus$	$R_\oplus$	6.371 010**E**6 *(5**E**-7)* m	Mean radius of the Earth
S_a	S_a	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.
S_b	S_b	6.356 752 314 2**E**6 *(1.6**E**-11)* m	Semi-minor axis of *WGS84*
S_{e^2}	S_{e^2}	6.694 379 990 14**E**-3 *(1.5**E**-12)*	First eccentricity squared of *WGS84*
$S_{e'^2}$	$S_{e'^2}$	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)
$S_{f^{-1}}$	$S_{f^{-1}}$	298.257 223 563 *(per definition)*	Flattening parameter of *WGS84*
T_0	T_0	273.15 K *(per definition)*	= 0°C, standard temperature

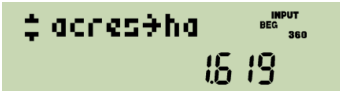
		Numeric value	Remarks
t_P	t_P	5.391 06**E**-44 *(6.0**E**-5) s*	Planck time $t_P = \sqrt{\frac{\hbar G}{c^5}} = \frac{l_P}{c}$
T_P	T_P	1.416 833**E**32 *(6.0**E**-5)* K	Planck temperature $T_P = \frac{c^2}{k}\sqrt{\frac{\hbar c}{G}} = \frac{M_P c^2}{k} = \frac{E_P}{k}$
V_m	V_m	0.022 413 968 *(9.1**E**-7)* 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	Z_0	376.730 313 461… Ω	Characteristic impedance of vacuum $Z_0 = \sqrt{\frac{\mu_0}{\varepsilon_0}} = \mu_0 c$
α	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	γEM	0.577 215 664 901 532 86…	Euler-Mascheroni constant γ_{EM}
γ_P	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	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	λ_c	2.426 310 238 9**E**-12 *(6.5**E**-10) m*	Compton wavelengths of the electron $\lambda_C = h / m_e c$, the neutron, and the proton, respectively.
λ_{cn}	λ_{cn}	1.319 590 906 8**E**-15 *(8.2**E**-10) m*	
λ_{cp}	λ_{cp}	1.321 409 856 23**E**-15 *(7.1**E**-10) m*	
μ_0	μ_0	1.256 637 061 4…**E**-6 $\frac{V\ s}{A\ m}$	Magnetic constant, a.k.a. vacuum permeability $\mu_0 := 4\pi \cdot 10^{-7} \frac{V \cdot s}{A \cdot m}$.
μ_B	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	μ_e	-9.284 764 30**E**-24 *(2.2**E**-8)* J/T	Electron magnetic moment
μ_n	μ_n	-9.662 364 7**E**-27 *(2.4**E**-7)* J/T	Neutron magnetic moment

		Numeric value	Remarks
μ_p	μ_p	1.410 606 743**E**-26 *(2.4**E**-8)* J/T	Proton magnetic moment
μ_u	μ_u	5.050 783 53**E**-27 *(2.2**E**-8)* J/T	Nuclear magneton $\mu_u = e\hbar/2m_p$
μ_μ	μ_μ	-4.490 448 07**E**-26 *(3.4**E**-8)* J/T	Muon magnetic moment
σ_B	σ_B	5.670 373**E**-8 *(3.6**E**-6)* $\frac{W}{m^2 K^4}$	Stefan Boltzmann constant $\sigma_B = \frac{2\pi^5 k^4}{15 h^3 c^2}$
Φ	Φ	1.618 033 988 749 894…	Golden ratio $\Phi = \frac{1+\sqrt{5}}{2}$
Φ_0	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	ω	7.292 115**E**-5 *(2**E**-8)* rad/s	Angular velocity of the Earth according to [WGS84](#) .
$-\infty$	$-\infty$	-Infinity	May the Lord of Mathematics forgive us calling these ‘constants’.
∞	∞	Infinity	
#	#		See the very last command in the index [above](#).

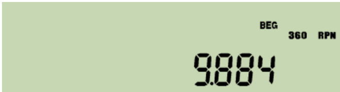
Unit Conversions (CONV)

CONV mainly provides the means to convert local to common units [57]. Navigation works as in the other catalogs. There is one specialty, however: f B (i.e. 1/x) will execute the inverse of the conversion displayed and leave CONV.

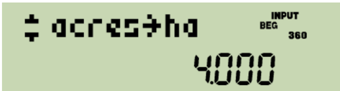
Example: Assume the display set to FIX 3. Then keying in

4 h CONV A will display 

telling you 4 *acres* equal 1.619 *hectares*.

Now press f B and you will get 

being the amount of *acres* equaling 4 *hectares*.

Press h CONV again and you see 

confirming what was just said.

Leave 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 light 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	$20 \lg\left({a_1}/{a_2} \right)$	Amplitude ratio	Ratio
atm→Pa	* 1.013 25E5		Pressure
AU→km	* 1.495 979E8	Astronomic units	Length
bar→Pa	* 1E5		Pressure

[57] For most readers, many of the units appearing in CONV may look obsolete at least. They die hard, however, in some corners of this world. All these corners have in common that English is spoken there. For symmetry reasons, we might also add some traditional Indian and Chinese units.

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 horse power	Power
hpUK→W	* 745.699 9	British horse power	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	/ 3.6**E**6		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	Astronomic 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 25**E**5	$1\ Pa = 1\ N/m^2$	Pressure
Pa→bar	/ 1**E**5		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	$10\lg\left(P_1 / P_2\right)$	Power ratio	Ratio
psi→Pa	* 6 894.757	Pounds per square inch	Pressure
PS(hp)→W	* 735.498 8	Horse power	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→HPe	/ 746		Power
W→hpUK	/ 745.699 9		Power
W→PS(hp)	* 735.498 8		Power
yards→m	* 0.914 4		Length

The constant $\mathbf{T_0}$ may be useful for conversions of temperatures, too; it is found in *CONST* and is not repeated here since being only added or subtracted.

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

3 0 h CONV P S XEQ
h CONV P ▼ XEQ resulting in 2,1bar.

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

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

In this area also some outdated units may be found in older literature: Pour les amis de Mme. Curie, $1Ci = 3.7 \cdot 10^{10} Bq = 3.7 \cdot 10^{10} decays/s$. And for those admiring the very first Nobel laureate in physics, Mr. Röntgen, for finding the x-rays (ruining his hands in these experiments), the charge generated by radiation in matter was measured by the unit $1R = 2.58 \cdot 10^{-4} As/kg$. A few decades ago, Rem (i.e. $Röntgen\ equivalent\ men$) measured what $Sievert$ 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 index above), there may be further labels employed and provided for particular tasks already. 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 website of *WP 34S* 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 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. Thus it will not be repeated here.

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 *below* about how to do this).

If you copy `wp34s-lib.dat` (some 3kB) 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 your calculator, a special connecting cable, and a specific software on your computer. It is beneficial if your computer features an hardware serial port.

- You will get the cable from Gene Wright (USA).

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

- 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 package you can download from here:

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

  Alternatively, you may download `calc.bin` or one of its siblings [58] alone from

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

  > **WARNING:** Flashing your *HP-20b* or *30b* will erase the *HP* firmware: your business calculator will be gone then. The *WP 34S* file will replace the *HP* firmware completely! Thereafter you will have a *WP 34S RPN Scientific* – i.e. your calculator will react as documented in this very 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 you (have to) interrupt the procedure at any time for any reason whatsoever, don't worry: simply start again.
  >
  > As long as the cable is connected to your calculator, it may draw a considerable current from its batteries. If you happen to hang anywhere in the flashing procedure, the processor of the calculator may be left running at full speed so your coin cells will be drained while you are trying to find out what is going wrong. Thus it is wise to disconnect your calculator when you will not need the cable for the next couple of minutes. For repeated flashing, an external 3V DC supply may pay very fast. Take care to connect '+' to the outer and '–' to the inner contact. The following procedure will work with a good 3V supply only.

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

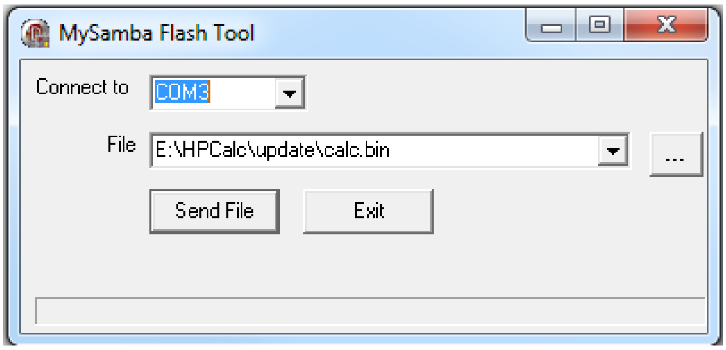
---

[58] These are the alternatives:
`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 needing the quartz plus an IR-diode built in,
and all files called `..._full.bin` include the flash memory user code library already – 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 you fulfill the hardware requirements or your calculator will hang.

Having prepared all the necessary items, follow the steps below:

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

2. Run *MySamBa*, and pick the serial port that you used for your cable and the file that you want to transmit. The window may look like this:



   Do <u>not</u> start sending yet! Just leave the window open.

3. Turn the calculator on by pressing and releasing ON/CE (the key bottom left).

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.

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 care – see the note above.

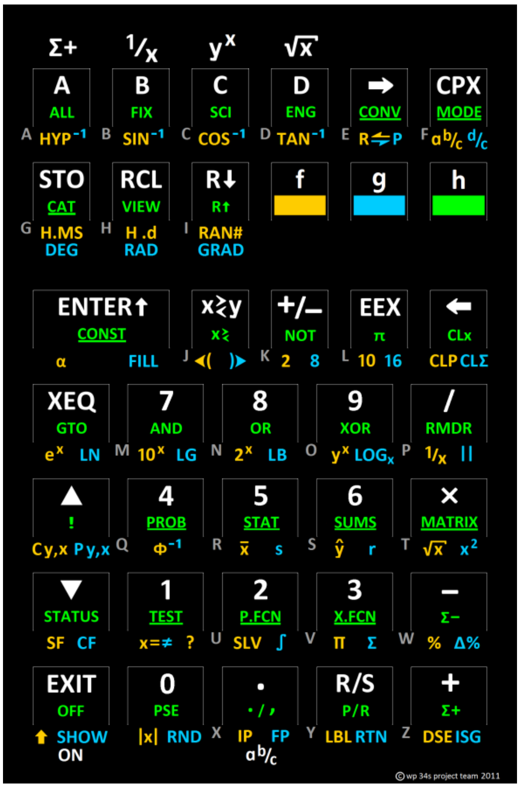
10. Click "Send File" in *MySamBa* and wait for it to finish transmission (it will only take a few seconds). If you have the FTDI USB/serial adapter, you will see the blue TX light blinking.

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.

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

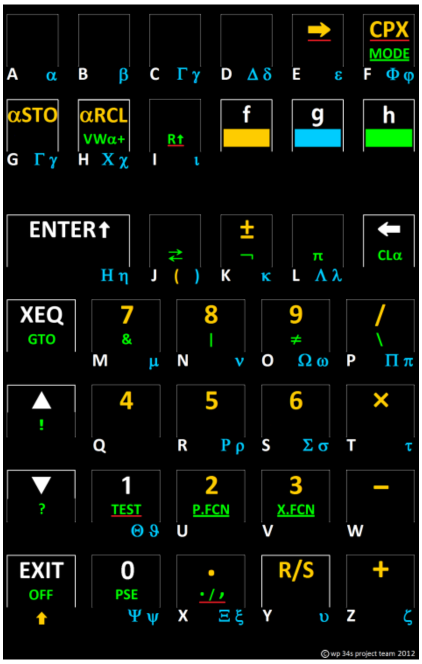


After flashing successfully, a keyboard overlay is very helpful for further work since most labels deviate from the ones used on said business calculators. You may get fine adhesive vinyl overlays from Eric Rechlin (USA). Supporters of the 'Great American Divide' can get it with a ÷ instead of the / as well.

If those vinyls are not available for any reason whatsoever, preliminary paper overlays are most easily made using the picture shown here at left. Print this page to scale (on A4 paper the picture shall be 68*mm* wide per default), cut it out, span it over your *WP 34S* using some transparent adhesive tape, and you are done.

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

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 some transparent adhesive tape. Or you may slide it 'under the hood' where the batteries live, as sort of a Quick Reference Sheet. The first 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 index of operations), you may use another approach. Hold down ON (i.e. EXIT) and press one of the following keys:

STO for backup:	Creates a copy of the *RAM* in flash memory like SAVE does.
RCL for restore:	Restores the most recent backup like LOAD does.
6 (i.e. **S**) for SAM-BA:	Clears the GPNVM1 bit and turns the calculator off. This will work in debug mode only (see below) to prevent accidental access to this possibly dangerous feature. **WARNING:** You can now only boot in SAM-BA boot mode! Without the SAM-BA software and the cable mentioned above, you are lost!

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

We recommend doing a SAVE or ON + STO before flashing a new release! After flashing, your backup will still be available – if you didn't accidently press the ERASE button on the cable but used ON + 6 instead to get into SAM-BA boot mode.

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

Mapping of Memory Regions to Emulator State Files

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

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

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

The following commands allow for sending programs, registers or all *RAM*. They are found in 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 <u>without</u> 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 chapter, 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 in four distinct sectors:

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

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

A complete copy of the nonvolatile *RAM* can be written to *FM* using SAVE (or ON+STO). 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 very top of available memory and is completely user transparent. Thus it is not covered here besides saying it contains 42 *bytes* of status and modes data, the alpha register, and 14 *bytes* holding the 112 global user flags.

Global Registers

Global registers are placed near the end of available memory. In startup default memory layout, the numbered registers **R00** to **R99** precede the stack and special registers **X, Y, Z, T, A, B, C, D, L, I, J,** and **K** as shown *above*. This totals to 112 global registers, which is the maximum available. Their number can be reduced down to the 12 special (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:** Please see the global register sector at startup default in the left three columns of the following memory table. The next two sets of two columns each show what happens after subsequent execution of REGS 96 and REGS 98. The registers are loaded with arbitrary values here to allow tracing them easily. *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**				
…						

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

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

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

Summation Registers

The memory needed for cumulative statistics is allocated separately – these data are no longer held in global 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 [59]. 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 *below*).

After CLΣ, CLALL, or RESET, the memory allocated for the summation registers is released again. 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) and after that load the registers would no longer fit in memory. You can avoid this by reducing the amount of numbered registers using REGS before the load attempt. This will (hopefully) move the summation data out of the way.

SRS and Program Memory

Both share the remaining space at lowest memory addresses.

The ***SRS*** is used for return addresses (sic!) 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, enhancing the flexibility of programs significantly. 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.

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

---

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

Making Room for Your Needs

The 12 special (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 max. 112 of them. So ***r*** varies between 48 and 896 (this maximum is explained *below*). Startup default is 448.
- ***s*** = words allocated for summation registers (70 if they are used, startup default is 0).
- ***p*** = number of words available for program steps and *SRS*. One step is taken by the inevitable final END statement already, 6 *words* are 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:

1. Reduce the number of global numbered registers allocated. One register less allows for four additional program steps typically.
2. Move programs to flash memory and clear the respective steps in *RAM*. Four cleared program steps allow for one additional register typically.
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 different 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 be also directly addressed using a dot heading the number – the arguments go from **.**00 to **.**15, corresponding to relative addresses from 112 to 127 [60]. 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.

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

RTN or POPLR – executed in a program – check if the current *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, else the pointer to the active local frame is cleared. RTN will branch to the return address found while POPLR will just continue execution. So the – until then – current local frame is dropped and the next higher (older) frame reactivated if existent.

Manually executing RTN, starting a new program with XEQ, 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 LastX correctly nor will it work for input greater than some hundred):

```
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 $\boldsymbol{r.00_3}$ there, returning to FAC_2 with $x = 2$. This x will be multiplied by $\boldsymbol{r.00_2}$ there, returning to FAC with $x = 6$, where it will be multiplied by ***r.00*** and will become 24 finally.

Switching between Real and Integer

Your *WP 34S* starts in standard real mode (DECM) when you get it new. You may use it for integer computations as well, as shown above many times. Going from DECM to any integer mode, the values on 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 subsequent 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	512,005 d	6,291,462 d
Recall the registers by sRCL			300 d	4,000 d	50,000 d	600,000 d
567 STO J, -9 STO 00	-9 d	567 d	567 d	40,964 d	-9 d	6,291,462 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	6,291,462.0

Note identical register contents are interpreted quite differently in DECM and integer modes. Even very small integers may induce very big surprises:

	R00	R01	R02
Contents at start e.g.	0	2	10
Then after WSIZE 64, BASE 16	0 h	2238000000000002 h	223C000000000001 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 → 0 1	10.	just for verification: OK
g 16	A h	turn to BASE 16
RCL → 0 1	Out of range Error	compare the table above
2 STO 0 1	2 h	
RCL → 0 1	000000000001 h	now this works in hex mode, too
f ◄	223C h	
f H.d	2.46684669589 18	turn to DECM
RCL → 0 1	0.	compare the table above – indirection has to ignore all fractional components. Otherwise ISG/DSE loops all simply would not work.

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

- Zero is stored as integer zero, i.e. all bits cleared.
- The mantissa (also known as *significand* in this context) is encoded in 5 groups of 3 digits. Each such group is packed into ten bits straight forward, meaning e.g. $555_{10} = 10\ 0010\ 1100_2$ or $999_{10} = 11\ 1110\ 0111_2$. 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 (64^{th}) 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 bits, the lowest 8 bits are reserved for the exponent. For the top 5 bits it is getting complicated now [61]: if they read …

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

- `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 bits will be the top bits of the exponent. If it 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 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
$9.99{\cdot}10^{99}$	23 bC 00 00 00 00 03 E7		0010 0011 1011 11	495
$1{\cdot}10^{-99}$	20 AC 00 00 00 00 00 01		0010 0000 1010 11	299
$1{\cdot}10^{-383}$	00 3C 00 00 00 00 00 01		0000 0000 0011 11	15
$9.999\ldots{\cdot}10^{384}$	77 FF E7 F9 FE 7F 78 00	9 3E7 3E7 3E7 3dE 000	0111 0111 1111 11	767

The second last number is the smallest one that can be entered numerically, the last one (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 $9.999{\cdot}10^{372}$ to the greatest number displays $1{\cdot}10^{385}$ stored as

$1{\cdot}10^{385}$	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, there are three special numbers featured by your *WP 34S*:

Infinity	78 00 00 00 00 00 00 00		0111 1000 0000 00	n/a
-Infinity	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 them 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 chapter about display already. 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
Bad time or dAtE	2	DECM	Invalid date format or incorrect date or time in input, e.g. month >12, day >31 etc.
Bad digit Error	9	Integer	Invalid digit in integer input, e.g. 2 in binary or 9 in octal mode. Will be displayed as long as the respective key is pressed.
Bad mode Error	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.
Domain Error	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 $\boxed{\text{CPX}}$), by $0/0$, $\Gamma(0)$, $\tan(90°)$ and equivalents, by $\operatorname{artanh}(x)$ for $\lvert\operatorname{Re}(x)\rvert \ge 1$, by $\operatorname{arcosh}(x)$ for $\operatorname{Re}(x) < 1$, etc.
Flash is FuLL	23	All	No more space in flash memory. Delete a program from flash to regain space.
Illegal OPErAtion	7	All	May appear in an attempt running an old program containing a command which turned non-programmable after said program was written.
Invalid dAtA	18	All	Set when there is a checksum error either in flash or as part of a serial download. It is also set if a flash segment is otherwise not usable.
Invalid PArAMEtEr	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 $j < 0$.
I/O Error	17	¬α	Please see Appendix A.

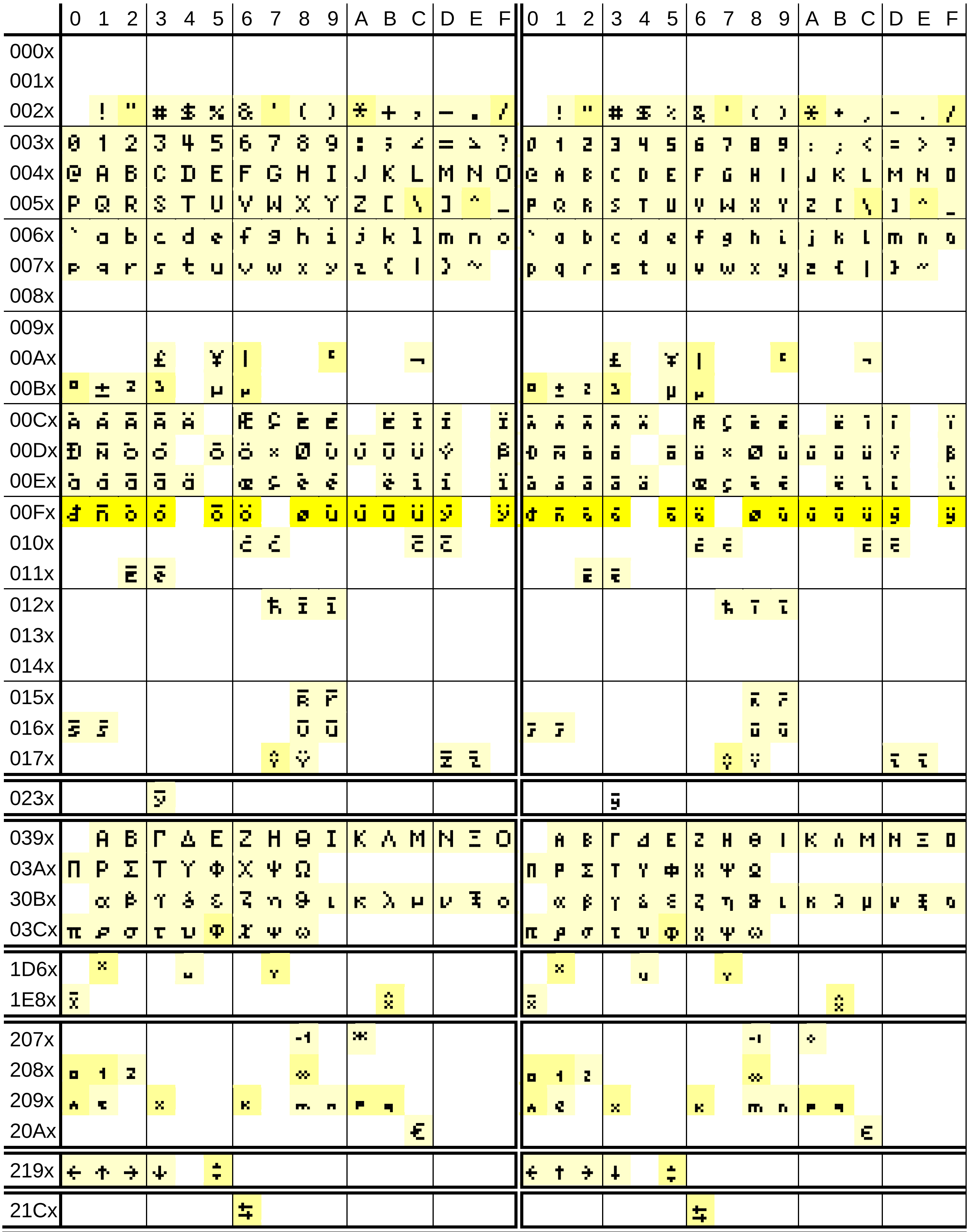
Message	Error code	Mode(s)	Explanation and examples
Matrix MISMATCH	21	DECM	• A matrix isn't square when it should be. • Matrix sizes aren't miscible.
No root Found	20	DECM	The solver did not converge.
No such LAbEL	6	All	Attempt to address an undefined label.
Out of range Error	8	All	• A number exceeds the valid range. Caused e.g. by specifying decimals >11, word size >64, negative flag numbers, integers $\geq 2^{64}$, hours or degrees >9000, invalid times, denominators ≥9999, etc. • A register address exceeds the valid range of currently allocated registers. May also happen in indirect addressing or calling nonexistent locals. • 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.
Too few dAtA Points	15	DECM	A statistical calculation was started based on too few data points, e.g. regression or standard deviation for < 2 points.
Too long Error	10	All	Keyboard input is too long for the buffer. Will happen e.g. if you try to enter more than 12 digits.
Undefined OP-COdE	3	All	An instruction with an undefined operation code occurred. Should never happen – but who knows?
Word size too SMALL	14	Integer, ¬PR	Register content is too big for the word size set.

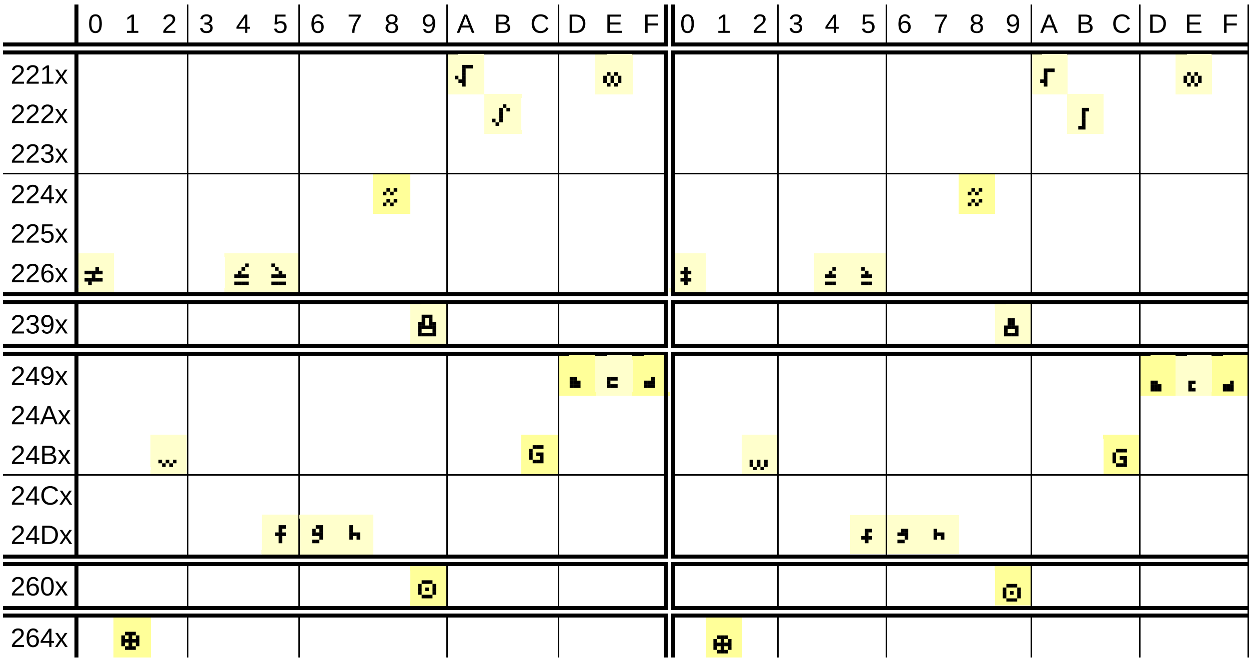
Message	Error code	Mode(s)	Explanation and examples
Write ProtEctEd	19	All	Attempt to delete or edit program lines in flash memory.
+∞ Error	4	¬α, ¬PR	• Division of a number > 0 (or < 0) by zero. • Divergent sum or product or integral. • Positive (or negative) overflow in DECM (see *above*).
−∞ Error	5		

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: CHARACTER SETS

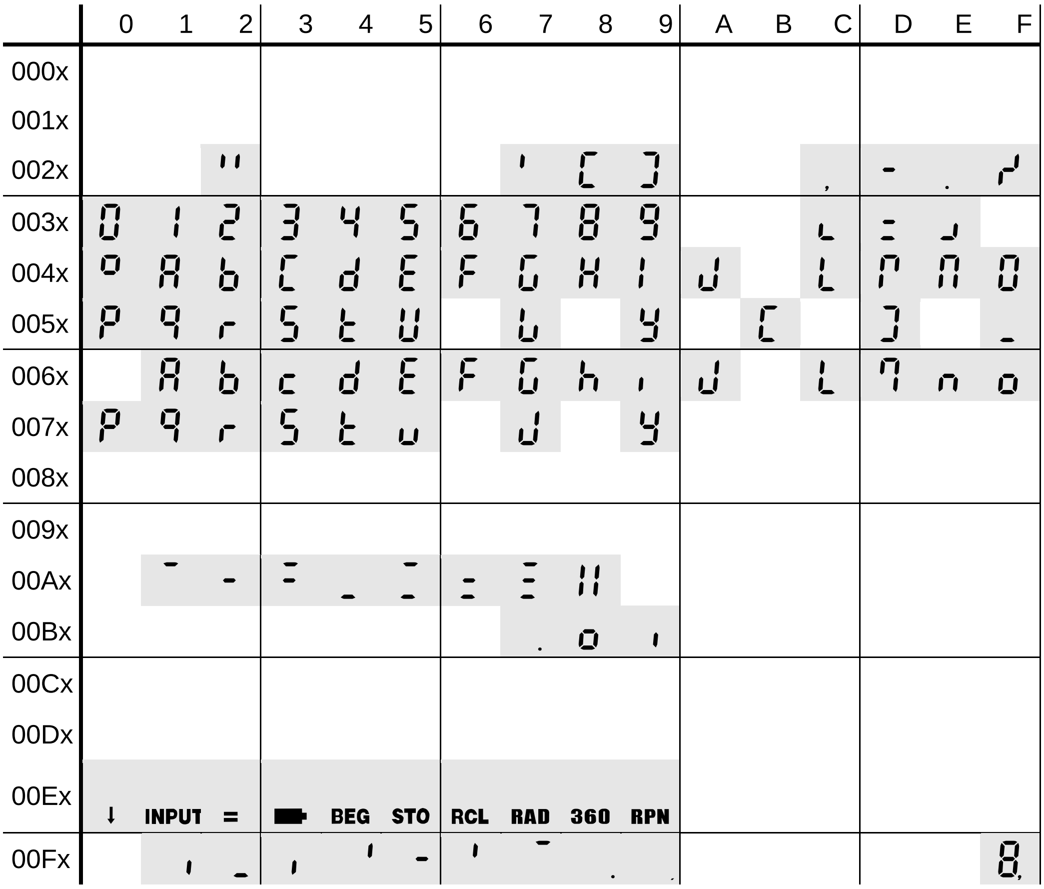
The following table shows the complete dot matrix character set as implemented in big and small font sorted according to the hexadecimal character codes (Unicode). In addition, there are two raster fonts supplied. Note the internal character sorting in your *WP 34S* differs from Unicode:





Characters from 00F0 to 00FF (printed on yellow) cannot be the last ones in a group of three.

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



All these fonts are available on the *WP 34S* website.

APPENDIX E: EVEN MORE COMMANDS

Commands for Advanced Users

The following operations ease some internal jobs, mainly in programming, but require special care and/or a deeper understanding of the respective 'mechanics' of the *WP 34S*. Else you may be found surprised by their consequences. **Use them at your own risk!**

They are not documented above but are collected separately in the expert's catalog (CPX h P.FCN , callable in DECM). These commands may also be generated by the assembler in translation. The table below follows the rules applying for the master index above.

Name	Keys to press	in modes	Remarks
BACK	… BACK ***n***	PR	(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, CASE, BSRB, or BSRF jumps, this may well result in a need to manually maintain all those statements individually.
BSRB	… BSRB ***n***	PR	(0) BSRB (BSRF) calls a subroutine starting ***n*** steps backwards (forwards) with 0 ≤ ***n*** ≤ 255. It pushes the program counter on *SRS* and executes BACK ***n*** (SKIP ***n***) then. The subroutines called this way do not require a starting label. So, BSRB and BSRF are most useful if you are short on local labels. See BACK for the price to pay.
BSRF	… BSRF ***n***		
CASE	… CASE ***s***	PR	(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 . It might exceed your expectations for ***r12*** > 4 , however, so take care!

Name	Keys to press	in modes	Remarks
CNST	… CNST ***n***	PR	(-1) Allows indirect addressing of the contents of CONST.
^{C}*CNST*	… ᶜCNST ***n***	PR	(-2) Works like CNST but does a complex recall. See *above*.
CNVG?	… CNVG? ***c***	DECM	(0) Checks for convergence by comparing x and y as determined by the lowest five bits of ***c***. The very lowest two bits set the tolerance limit: 0 = 1E-14, 1 = 1E-24, 2 = 1E-32, 3 = Choose the best for the modes set, resulting in taking 0 for single precision and 2 for double precision. The next two bits determine the comparison mode: 0 = compare the real numbers x and y relatively, 1 = compare them absolutely, 2 = check the absolute difference between the complex values $x + i\,y$ and $z + i\,t$, 3 = works as 0 so far. The top bit tells how special numbers are handled: 0 = NaN and infinities are considered converged, 1 = they are not considered converged.
		Integer	(0) Tests for $x = y$.
DBLOFF	… DBLOFF	¬α	(0) Toggles double precision mode. Setting becomes effective in DECM only and is indicated by D in the dot matrix. See next paragraph.
DBLON	… DBLON		
DBL?	… DBL?	¬α	(0) Checks if double precision mode is turned on.
dRCL	… dRCL ***s***	¬α	(-1) Assumes the source ***s*** contains double precision real data and recalls them as such. See next paragraph.
RTN+1	… RTN+1	PR	(0) Works like RTN, but moves the program pointer two steps behind the XEQ instruction that called said routine. Halts if there is none.
SKIP	… SKIP ***n***	PR	(0) Skips ***n*** program steps forwards (0 ≤ ***n*** ≤ 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. Compare BACK.

Name	Keys to press	in modes	Remarks
⇆	… ⇆____	¬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 also ⇆ZZZX is possible. Play! But remember it will not affect the higher levels in an 8-level stack.

ON + C : Tells the system a quartz **c**rystal is installed for the real time clock. This hardware modification (described in `How to install crystal and IR diode.pdf` in the folder 'doc') is an inevitable prerequisite for the clock being useful in medium to long range (see TICKS, STOPW). It is required for the print operations, too.

> **WARNING:** If ON + C is entered although the hardware does not contain said modification, the system will hang and can only be brought back to life by a hard reset or a battery pull! – You will reach the same hang state if you attempt booting a firmware requiring the quartz installed (see *above*) though you did not do that before.

ON + D : Enters **d**ebugging mode.

Double Precision Calculations and Mode Switching

Your *WP 34S* starts in standard (i.e. single precision or *SP*) real mode per default. Switching between *SP* and integer modes was discussed *above* already. 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). Please note matrix commands will not work in *DP*.

DP allows for more precise calculations. While some calculations will reach high accuracy, we do not warrant, however, 34 digit precision throughout your *WP 34S* in *DP* mode. Please keep this in mind always.

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 cut into the former program sector then.

Returning from *DP* to *SP* mode, the lettered registers are copied again. And 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 containing zero now.

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 [62]	3.43 -395	1.40 -397	1.03 -HIG	1.95 -HIG	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 -HIG	1.95 -HIG	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

Please note iRCL and sRCL keep working as explained above.

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

314159 DEG 360
2653589793 0 000 , and ▶

238462 DEG 360
643383279503 .

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

[62] *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 as in *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 $10^{-6143} \le |x| < 10^{+6145}$. 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 $10^{-999} \le |x| < 10^{+1000}$ will be displayed with -HIG or HIG in the 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*** = 1.951 656 000 000 000 000 543 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.

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 its preprocessor already) will translate most of them into proper program steps employing the commands documented above. Please see the *WP 34s Assembler Tool Suite User Guide* (`.pdf`) for additional information.

<table>
<tr><th>(Pseudo-) Command</th><th>XROM or LIB</th><th colspan="2">Function</th></tr>
<tr><td>A..D→</td><td>XROM</td><td colspan="2">saves a, b, c, and d at a volatile temporary storage. Volatile means it will vanish some 500ms after last keystroke.</td></tr>
<tr><td>GSB label</td><td>Both</td><td>This pseudo-command allows for calling long alphabetic labels. The assembler translates GSB into BSRB or BSRF – or XEQ (via a jump table in XROM or via a label in LIB) if the distance is too far.</td><td rowspan="2">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 labels looking like label:: may be in input to the assembler, but will be resolved.</td></tr>
<tr><td>JMP label</td><td>Both</td><td>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.</td></tr>
<tr><td>Num sn</td><td>XROM</td><td colspan="2">with sn = 0, π, √2π, … inserts the respective constant like recalling from CONST would do.</td></tr>
<tr><td>POPUSR</td><td>XROM</td><td colspan="2">See XEQ or XEQUSR.</td></tr>
<tr><td>XEQ label</td><td>XROM</td><td colspan="2">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.</td></tr>
<tr><td>XEQUSR label</td><td>XROM</td><td colspan="2">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..</td></tr>
</table>

(Pseudo-) Command	*XROM* or *LIB*	Function
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.

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.

APPENDIX F: RELEASE NOTES

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

1.14	18.3.11	Started the Windows emulator. Added DEC and INC, renamed FLOAT to DECM; redefined αTIME and H.MS mode; updated appendix A; documented the annunciators BEG and = as well as underflows and overflows in H.MS; corrected some errors showing up with the emulator.
1.15	21.3.11	Modified FIX, removed ALL from MODE, updated CONV.
1.16	27.3.11	Added LBL?, f'(x), and f"(x); modified PSE; upgraded catalog searching.
1.17	9.5.11	Modified keyboard layout for adding a fourth hotkey; added AGM, BATT, B_n, B_n*, Cauch, Lgnrm, Logis and their inverses, all the pdf, COV, CUBE, CUBERT, DEG→, ENGOVR, ENTRY?, erfc, GRAD→, GTO . *hotkey*, KEY?, RAD→, SCIOVR, SERRw, SLVQ, sw, sxy, TICKS, TVM, xg, ε, ε_m, ε_p, ζ, σw, $(-1)^x$, the polynomials, four angular conversions, four Planck constants, the regional settings, global alpha labels, and three messages; renamed most cdf; changed →DEG, →RAD, →GRAD to leaving angular mode as set; altered PSE for early termination by keystroke; made D.MY default instead of Y.MD; moved degrees to radians conversions to CONV; removed ^{C}CLx, H.MS mode, %+ and %-; corrected errors.
1.18	5.6.11	Expanded program memory; modified label addressing (A ≠ 'A') and fraction mode limits, changed ANGLE to work in real and complex domains, renamed MOD to RMDR, changed the keyboard layout; put BACK, ERR, SKIP, and SPEC? to the main index; added CAT and the I/O commands for flash memory, expanded R-COPY; corrected x→α.
2.0	21.7.11	Entered beta test phase. Added DAY, MONTH, YEAR, FAST, SLOW, S.L, S.R, VWα+, flag A, ON + and –, some constants, and a paragraph about I/O; renamed old DAY to WDAY, RRCL to RCFRG, SRCL to RCFST; added an inverse conversion shortcut, stones⇆kg, and changed Pa⇆mbar to Pa⇆bar; modified the VIEW commands, ALL, DISP, MODE, RCLM, STOM, and X.FCN; repaired hyperlinks; corrected some errors; included flash.txt; updated the first chapters, explained stack mechanics in more detail.
2.1	3.10.11	Added serial I/O commands, DEL_P, DSL, EXPT, IBASE?, INTM?, ISE, KTY?, MANT, NEXTP, PUTK, REALM?, RM, RM?, SMODE?, TOP?, $\sqrt[x]{y}$, 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	29.5.12	Added the print commands 🖶…, error probabilities, $I\Gamma_q$, XTAL?, γ_{XY}, Γ_{XY}, an optional back label, and some references to external documents; renamed IΓ to $I\Gamma_p$; moved BACK, RTN+1, SKIP, and ⇆ to internal / expert commands – relocated this catalog to key 2; updated and rearranged large parts of the introductory text, updated App. A and C, and expanded App. B and E, also explaining the floating point formats.