

Figure 1 – Keyboard Navigation Summary Overview

(1) The top line of keys can be used to quickly execute a program label when any of the labels A, B, C or D are defined within a program by the user. If these are not defined, then the primary function of each key will be sigma add, reciprocal, power or root respectively. Note that each of these functions is available elsewhere on the keyboard.

(2) Enters a number.

(3) The primary function of the XEQ key is to execute a function or program

(4) These scroll keys are used to move up or down within a program or a catalogue list

(5) The primary function of the EXIT key is a general escape or cancel button. Secondary functions are used to switch the calculator on/off. Yellow shift switches between upper and lower case when entering text and blue shift "SHOW" can be used to view any register (including the stack contents).

(6) Primary function of this key is to run/stop a program. Secondary functions include inserting labels, or subroutine returns.

(7) Backspace key, with secondary function used to clear x, clear program or clear sigma data.

(8) Every Keys primary function is shown in white on the front face of the keys. Secondary functions are shown as yellow, blue or green requiring a prefix of these respective f, g or h keys. Keys are used when entering text characters – and these are shown as grey characters (shown at the bottom left of the key). For example the EEX key is pressed when entering the letter "L"

(9) The CPX (complex) primary function is used as a prefix for commands that must execute in the complex domain. Pressing CPX before + for example would perform a complex math addition. Observe that the green secondary function of this key has an underline (MODE) which means that this key opens a catalogue of functions (in this case related to the calculator mode of operation).

(10) The right arrow key is used as a prefix to force a conversion of whatever is in the X register. It can be used before H.d, H.MS, DEG, RAD, GRAD. It can also be used before the bases "2" and "8" (+/- key) and "10", "16" (EEX key) to perform quick base conversions. Additionally this button is pressed to signify the use of indirect addressing mode.

# **WP 34S**

## **RPN Scientific Calculator**

### **Beginners Guide**

*Front cover: WP 34S by Walter, Pauli, Marcus, Neil, Eric and many others, accompanied by a much loved HP 32E in the background*

*Dedicated to my wife who also helped me see the light*

## Index

Index .....	5
Table of Figures .....	9
Draft Release Change Details.....	10
If you ever find yourself in an unexpected mode? .....	12
Who am I? .....	12
Introduction .....	13
Where this document fits in and the inevitable legal disclaimer .....	15
Getting Started.....	17
Switching On .....	17
Setting the LCD Display Contrast.....	18
Entering numbers.....	18
Changing the sign of the number .....	18
Clearing .....	18
Functions using one number .....	19
Functions using two numbers.....	21
Chain calculations .....	23
Setting the display format.....	24
Scientific display notation - SCI.....	24
Engineering display notation - ENG.....	26
Fixed display notation - FIX .....	27
Fixed display notation – Engineering / Scientific Override .....	27
Setting the number of display digits - DISP .....	28
All display notation - ALL.....	29
Viewing all digits in the Mantissa .....	30
Setting the angular format .....	31
Entering Exponents .....	31
Catalogues .....	33
Using catalogues to set calculator preferences .....	33
Setting UK preferences.....	34
Quickly locating catalogue functions .....	35
Quickly locating catalogue functions with Greek letters .....	35
Fixing the display to four decimal places.....	35
Setting the Time .....	36
Setting the Date.....	36
Reading the Time from the calculator.....	37
Reading the Date from the calculator .....	37
The Automatic Memory Stack .....	38
Initial Display .....	38
Stack control - manipulation.....	39
Roll Down (and Roll Up) Key.....	39
Exchanging X and Y .....	41
Clearing the Stack .....	42
The ENTER Key .....	42
Stack Arithmetic .....	44
Chain Arithmetic .....	45
Constant Arithmetic.....	48
Order of Execution .....	49
LAST X.....	50
Complex domain LAST X .....	51
Memory Organisation .....	52
Global registers.....	53
Global Register Access – Direct and Indirect .....	54
Global Register Access – Out Of Range Error.....	55

Cumulative Summation Registers – Sigma Data .....	55
Program Steps & Subroutine Return Stack.....	57
Local Data .....	57
Clearing memory to make space .....	58
Backing up the calculator memory - introduction .....	58
Backing up the calculator memory – hot key shortcut .....	59
Creating a new backup using ON & STO .....	59
Restoring a backup using ON & RCL .....	59
How STO (+,-,/,x) and RCL (+,-,/,x) access items in memory .....	60
STO and RCL register access - summary .....	61
How tests such as x=? access items in memory .....	61
Tests x=? register access summary .....	62
How simpler commands such as FIX access items in memory .....	62
Temporary Alpha Layout .....	63
Integer modes - using alternate bases .....	65
Complement and unsigned mode .....	68
1's Complement Mode .....	68
Viewing long number displays – introduction .....	69
Viewing very long 64 bit binary numbers.....	71
Leading Zero's ON / OFF switch.....	73
2's Complement Mode.....	74
Unsigned Mode.....	74
Bit wise operations and integer math.....	75
Integer arithmetic functions.....	77
Addition, Subtraction, Multiplication and Division .....	78
Addition and Subtraction in 1's complement mode .....	79
The carry flag during addition .....	80
The carry flag during subtraction .....	80
Overflow – WP 34S flag B .....	81
Remainder after division and RMDR .....	82
Square root .....	82
Negative Numbers – changing signs.....	82
Negative Numbers – absolute value .....	82
Logical operations .....	83
Logical NOT .....	83
Logical AND .....	83
Logical NAND .....	84
Logical OR .....	84
Logical NOR .....	85
Logical XOR .....	85
Logical XNOR .....	86
Bit Shifting and Rotating .....	87
Shifting Bits.....	87
Logical Shifts .....	87
Left and Right bitwise Justification .....	88
Arithmetic Shift Right .....	89
Rotating bits .....	89
Rotation.....	90
Rotation through the carry flag .....	90
Negating, Asserting and testing the state of bits .....	91
Testing bits.....	91
Bit masks – left and right justified .....	92
Mirroring bits .....	92
Double Functions.....	92
Double Multiply - DBLx.....	93
Double Divide – DBL/ .....	94
Double Remainder .....	95

Programming.....	97
Keying in our first program – ‘CIR’ .....	98
Program Pointer .....	102
Program-Entry mode .....	102
Program-Step Number .....	102
Inserting new program steps .....	102
Deleting existing program steps .....	102
Branching and Calling – what’s the difference?.....	103
Program Labels .....	103
Global Labels .....	103
Local Labels .....	104
Modifying ‘CIR’ to demonstrate global AND hotkey local labels .....	104
Predefined local program labels A, B, C, D – hotkey labels.....	107
Key Code Labels .....	107
Number labels – 00 to 99 .....	110
Direct program execution using number labels.....	111
Indirect program execution using number labels .....	112
Indirect addressing – used to access the stack.....	113
Deleting a specific program .....	114
Deleting all programs.....	114
Saving Programs to the Flash RAM Library - advanced .....	114
Program Input / Output .....	115
The Alpha Register – how to display text.....	116
Inserting special characters into alpha .....	118
Alpha Catalogues.....	118
Common math symbol alpha catalogue .....	118
Common math symbols super/subscripted alpha catalogue .....	119
Punctuation symbol alpha catalogue .....	120
Math symbols (stats & complex domain) alpha catalogue .....	121
Arrows and extra math symbol alpha catalogue .....	122
Other alpha characters obtained direct from the keyboard .....	122
English alpha characters (A-Z, a-z) .....	122
Number digits (0 to 9) .....	122
Logic symbols .....	123
Other useful h shifted misc symbols – including a space.....	123
Other useful f shifted math symbols .....	123
Greek character symbols .....	124
Alpha Register Commands.....	125
Parallel Resistor Program using static Alpha displays.....	126
Archiving the program to the flash RAM library.....	130
Debugging the parallel resistor program .....	131
Simple Counting Program – showing a dynamic Alpha display .....	134
Counting Program illustrating Loop Control .....	136
Local Data - Variables .....	138
Local Data - Flags.....	139
Local Data - limitations .....	139
Fractions.....	143
Fraction modes – how the display works .....	144
Exact fraction conversions .....	144
Inexact fraction conversions .....	145
Accuracy Indicators .....	145
Fraction Mode – Rounding .....	147
Fraction Mode – Limitations .....	147
Larger fractions.....	147
Controlling the fraction display format .....	148
Setting the maximum denominator.....	149
Setting the default fraction mode – max precision .....	149



Setting the fraction mode to employ factors.....	149
Setting a fixed fraction denominator .....	149
Using fractions in programs .....	149
Complex modes of operation - introduction .....	151
Complex mode - extra function catalogue.....	152
Complex number format .....	152
Entering complex numbers and invoking complex functions .....	153
The complex stack.....	154
Complex Operations .....	155
Operation on one complex number .....	155
Operation on two complex numbers.....	156
Unit conversions - introduction .....	158
Immediate Conversions .....	158
Functional based conversions.....	159
Accepting a Conversion .....	160
Combining Conversions .....	160
Quick conversion inversion .....	161
Constants – Introduction.....	163
Use of constants in the complex domain.....	163
Statistical Functions - Accumulations .....	165
Deleting and Correcting Data .....	165
Finding the Mean.....	165
Standard Deviation.....	166
Linear Regression.....	168
Linear Estimation .....	170
Correlation Coefficient.....	170
Normal Distribution .....	171
Factorial .....	174
Percent of sum $\% \Sigma$ .....	175
Percentage Difference $\Delta \%$ .....	176
Hyperbolic Functions .....	177
Calculator Hardware.....	180
Replacing the batteries .....	180
Hardware reset .....	181
Calculator Firmware .....	182
What version firmware is my calculator using? .....	182
Upgrading firmware .....	182
Subject Index .....	183



## Table of Figures

<i>Figure 1 – Keyboard Navigation Summary Overview.....</i>	<i>2</i>
<i>Figure 2 – HP 42S side-by-side with a WP 34S.....</i>	<i>13</i>
<i>Figure 3 – Primary and Secondary key functions .....</i>	<i>17</i>
<i>Figure 4 – Table illustrating basic math two value operators .....</i>	<i>21</i>
<i>Figure 5 – Keyboard key codes for all keys.....</i>	<i>108</i>
<i>Figure 6 – Use of local data within nested functions.....</i>	<i>141</i>
<i>Figure 7 – A Mixed fraction.....</i>	<i>143</i>

## Draft Release Change Details

Release	Draft	Description
23 <sup>rd</sup> April 2012	2.06812	Base document – firmware version 3.0 2676
26 <sup>th</sup> April 2012	2.06813	<p>Added new sections:-</p> <ol style="list-style-type: none"> <li>1. Statistical functions accumulations</li> <li>2. Factorial (!)</li> <li>3. Percentage of sum (%Σ)</li> <li>4. Percentage difference (Δ%)</li> <li>5. Hyperbolic functions</li> </ol> <p>Also added a note in the section giving examples of one-function operations to demonstrate the 2<sup>x</sup> command</p>
27 <sup>th</sup> April 2012	2.06814	Extended the alpha register description section to illustrate how to obtain alpha math, logic and punctuation symbols, and also Greek characters. The information also shows the mapping between English and Greek letters.
28 <sup>th</sup> April 2012	2.06815 and 2.06816	<p>Extended the alpha characters listings – to show screen grabs of all characters (excluding English letters and numbers). The Greek section shows a properly ordered, full set of upper and lower screen grabs. Note this includes the missing Psi, Theta, Eta and a note regarding Omicron.</p> <p>Rationalised the indexing of the alpha section to split it into catalogues and direct key characters. Moved both indexes, and change history table using page breaks. Indexing added to PDF shell, cut &amp; paste content enabled. Added web link to email link in footer.</p>
1 <sup>st</sup> May 2012	2.06817	Added reference to the cpx h mode catalogue. Added section providing information on changing LCD contrast, and also creating and restoring backups from flash ram. This draft also provides a very small amount of information regarding archiving programs to the flash RAM library.
2 <sup>nd</sup> May 2012	2.06818	<p>For this draft – the test calculators have been flash upgraded from 3.0 2676 to firmware release 3.0T 2830.</p> <p>Corrects a mistake in the examples for string formats used by the SETTIM function, and the two minor errors in the DBL/ command explanation. It also adjusts the example in fractions dealing with the accuracy annunciation and shortens the introduction.</p>
9 <sup>th</sup> May 2012	2.06819	For this draft – the test calculators have been flash upgraded from 3.0T 2830 to 3.1 2932 and changes made to the fonts used in some of the examples. Also expanded the hardware section to show how version information is displayed, and to provide information on re-flashing.
25 <sup>th</sup> May 2012	2.06820	<p>This version contains the following changes</p> <ol style="list-style-type: none"> <li>1. New front cover with nods to older HP guides (many thanks Jeff Dinkins)</li> <li>2. Typo/spelling corrections (including the misspelling of Marcus – please accept my humble apologies).</li> <li>3. Extra explanation notes regarding the use of temporary alpha mode when using the STO, RCL and test catalogue functions.</li> <li>4. The automatic memory stack diagrams and the bitwise logic op diagrams made smaller.</li> <li>5. Key stroke sequences showing multi-digit entries (previously shown using one box per digit), now use one box for the number for brevity.</li> <li>6. Catalogue function name boxes now standardised in look.</li> <li>7. Added extra notes regarding the use of complex constants</li> <li>8. Mention made of move of the small advanced user catalogue</li> <li>9. Notes added re: best use of up/down arrow when locating catalogue functions.</li> <li>10. A slight <i>reduction</i> in accuracy has now been built into the memory section (to allow for changes that will inevitable occur in the future).</li> <li>11. The serial interface notes (Atmel processor) have been redrafted</li> <li>12. The display mode section (in particular ALL and FIXed modes) have been completely re-written in line with Pauls explanation – with worked examples.</li> <li>13. Wording changes referring to I, L, J and K registers in the memory section.</li> <li>14. The addition of a subject index at the end; created via concordance file.</li> <li>15. The format of the special blue "NOTE" areas of text has been standardised.</li> </ol> <p>This version was drafted after publically announcing the guide to the HP Museum forum.</p>

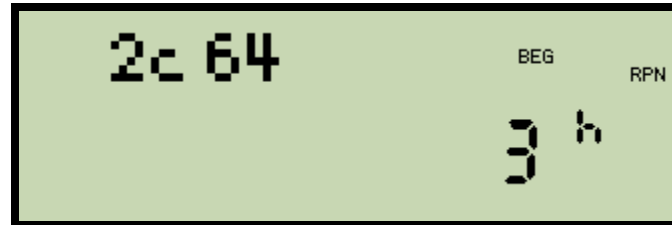
# Part 1

## WP 34S

### Introduction

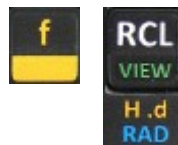
## If you ever find yourself in an unexpected mode?

The WP 34S can work in a number of different modes for example floating point or integer and then in any of a number of different bases. You might switch modes, and end up (as I did) with a display such as the following:-



Seeing this display means that you have switched the calculator to integer mode. The "h" indicates it is running base 16 (hexadecimal). The "2c" means it is working in 2's complement mode, and the 64 means it is using 64 bits of precision. The "BEG" means that the program counter is set to point to the beginning of program memory, and the 3 is (as you'd expect for an RPN calculator) simply the contents of the X register.

If you were ever to find yourself with this display, press the yellow "f" shift key, and then press the H.d key (which is a secondary function of the RCL key) to switch back to normal floating point mode.



...and if you ever need to cancel a command mid way through, just click the EXIT key (bottom left)

## Who am I?

First I am not involved in the project to design the WP 34S calculator in any way shape or form. Second I am not linked to Hewlett-Packard in any way – other than by over 30 years of use of their fine calculator machines (and for marginally less time, their printers).

I happened to stumble across the WP 34S project while searching the web for information on a totally different HP calculator (the limited edition HP-15C machine), and after reviewing the available WP 34S information felt it was rather too interesting to ignore. I obtained a WP 34S from Eric and then started trying to use it and that's when the fun started. Personally I found the documentation available so difficult to use that I decided to write my own self help guide. This document was the result. It isn't complete, and it isn't perfect, it might even be wrong but it may perhaps act as a starting point for a novice.

I am a software developer, who works mainly on back office web site software, but with both a professional and personal interest in embedded hardware and software design. I am rather interested in fuel injection systems and the control and computing systems they require. I live in the UK with my wife (and our small collection of HP calculators).

## Introduction

Scientists and engineers are a very discerning lot and (like myself) are loyal followers of Hewlett Packard who for many decades have specialised in designing a particular style of calculator employing an efficient postfix notation known as reverse polish notation (RPN). Historically the scientific and engineering market has always been closely linked to HP products, across whole families of RPN calculators stretching back to the late sixties and all characterised by the lack of an equal's key. We'll look at RPN later, but in short it has the unique advantage (compared to algebraic infix notation calculators) of avoiding the need for parenthesis which is both cumbersome to use and error prone. As a student, learning RPN had the added benefit of teaching me to *think* about the problem I was trying to solve – which in turn led to the benefit of both being able to estimate and constantly check my results before ending up with a number in a calculator display. The fact that you can see and use intermediate results easily when using RPN is perhaps its chief asset.

Once RPN is mastered, I doubt many would choose to go back to algebraic entry.

Scientists and engineers have been yearning for a successor to arguably the best Hewlett Packard calculator ever made – namely the HP 42S. The 42s was built as a successor to the fine HP 41CV and although the two were software compatible, the newer machine did not possess the same degree of hardware expandability. That said, it was smaller, had a much longer battery life and a two line LCD display facilitating an intuitive menu based user interface.



Figure 2 – HP 42S side-by-side with a WP 34S

Released in 1988, but discontinued in 1995 these fine calculators routinely sell for two to three hundred pounds second hand on eBay. They are an object lesson to HP that there is still significant demand for high end, non-graphing scientific calculators...

The WP 34S machine is an open source project designed to address this market need. The history of the project is relatively well documented online and doesn't need to be repeated here other than to summarise that the project started around 2004 and built on code originating from the "OpenRPN" ideal employing a community approach to developing RPN hardware and software and which eventually led to a collaboration between a group of like minded people (Walter, Paul, Marcus, Neil, Eric and many others) with the intention of developing what might be thought of as a HP-42sII. Early revisions extended the test systems by using existing HP calculators re-flashed with new firmware.

These started with a machine based on the existing HP 20b model but culminated in a project to reuse the HP 30b model (a financial calculator) with the following specifications

1. Two line display
  - a. 400 segments (12x9 mantissa, 3x7 exponent and 2 signs).
  - b. 6x43 dot matrix
  - c. And a set of 11 individual annunciators
2. Atmel CPU (AT91SAM7L128)
3. Serial port for CPU reflashing. The connector employed is located behind the battery cover, and has a six pin interface (serial transmit, serial receive, ground, Vcc along with reset and erase strobes). This interface adheres to an Atmel specific protocol for both erasing and flashing the CPU firmware. There is an additional JTAG connector on the PCB (a set of solder pads) but which is not accessible outside of an unmodified plastic calculator housing.
4. 37 Key keyboard (6 columns, 7 rows) – using HP's rotate and click key design.

**Note**



*The "rotate and click" keyboard design is an important selling point for any experienced HP user. Most users are attracted to HP calculators because of RPN, the sophistication of the software and the quality, durability and feel of the roll and click keyboards.*

By contrast, the HP 30b keyboard is a good standard and while not quite as good as some HP machines it is vastly better than either the 49g or 50g.

The processor in the HP 30b is a good platform upon which to base the re-purposed machine because it is quick and the hardware modifications required to support the change are reasonably straightforward. The unit has to be disassembled, a 32768Hz crystal and some surface mount C's added to the PCB to serve the real time clock (RTC). The unit is then reassembled and the CPU flashed using a serial port built into the case inside the battery compartment. At that point, the old HP 30b and all its financial software is no more. The final task remaining is to place a good quality printed vinyl overlay over the top of the keyboard and key heads. While the overlay isn't perfect and may wear with time it is very usable and of impressive quality.

Voila, a WP 34S is born.

### ***Where this document fits in and the inevitable legal disclaimer***

This is first and foremost a beginner's guide. It has been written to provide a basic working knowledge of the calculators operation. It is not meant to stray into the territory of what might be classed as advanced use - nor does it define the complete list of WP 34S functions built into the machine, nor the constants, nor the conversions available, nor the full list of catalogues. Paul and Walters manual covers all of those more formal definitions for the WP 34S.

#### **Note**



*This document is not an endorsement of the WP 34S machine and it makes no claim that the WP 34S is fit for any purpose whatsoever. I make no claim that this beginner's user guide is fit for any purpose whatsoever. If you choose to use either the WP 34S or this beginner's guide document then you do so entirely at your own risk.*

*Don't blame me if something goes wrong.*

If however this guide helps you on your journey, consider it a bonus which cost you nothing to enjoy.

Good luck and enjoy the machine!!



A large teal rounded rectangle with a thin red border, serving as a background for the section header.

# Part 2

## WP 34S

### Getting Started

## Getting Started

Most keys on the keyboard perform multiple functions. The primary function of a key is always shown as white text on the main face of the key (the one exception being the EXIT or ON key. When the calculator is running, this key has the primary function of acting as a generalised escape or cancel function key. However, when the calculator is switched off, the primary function is to switch the device on).

Other key functions are colour coded, and require one of the three colour shift keys (yellow "f" key, blue "g" key or the green "h" key) to be pressed prior to using. Lastly, some keys can be used when the calculator is in alpha text mode, in order to enter specific alpha characters.

A good example is the digit 7 key which looks as follows. The primary function of this key is shown in point (1). Secondary functions are shown in points (2) through (4) and the alpha text character is shown in point (5).

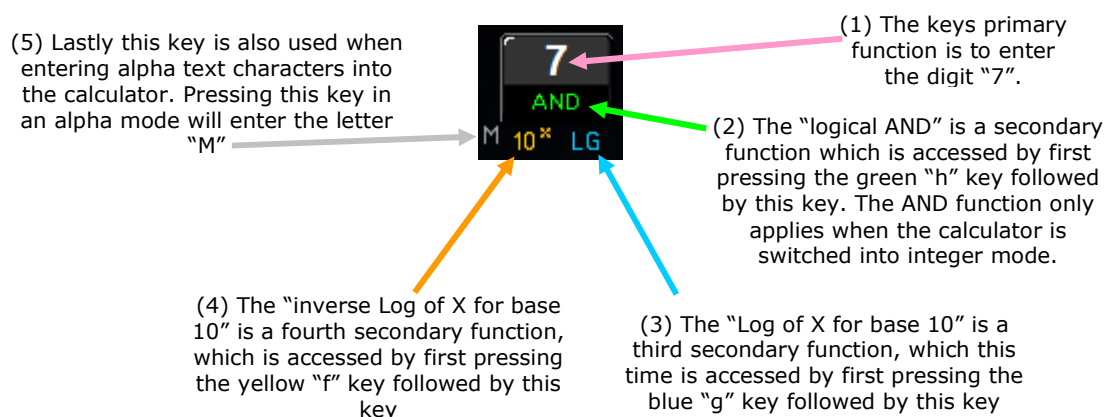
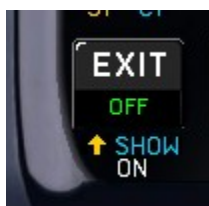


Figure 3 – Primary and Secondary key functions

## Switching On

To switch the WP 34S ON simply press the bottom left hand key – which is marked EXIT (it shows the word "ON" underneath).



To switch the calculator off, press the green "h" key, and then the "OFF" key, which is a secondary function of the EXIT key. We could show this pictorially as:-



The calculator will automatically power off after a period of inactivity. Early revisions of the firmware used a very short timeout but later revisions extended the timeout to 10 minutes.

## ***Setting the LCD Display Contrast***

You can adjust the display contrast by pressing two keys at the same time.

To change the display contrast press and hold the EXIT key and then at the same time repeatedly press either the minus key (-) to reduce, or the plus (+) key to increase. The display will show the annunciation word "Contrast" with either a prefixed minus or plus and the display will change accordingly.

## ***Entering numbers***

Enter numbers by pressing each digit in sequence just as they would appear on paper, inserting a decimal point when required. Note that if the calculator is working in UK mode (see Setting Preferences section below) then commas will be automatically inserted for you.



## ***Changing the sign of the number***

To change the sign on the number, press the +/- key



## ***Clearing***

You can either use the backspace key to clear one digit at a time, or you can prefix the backspace key with a green "h" shift to clear the entire X register.

This is one (of potentially many) secondary functions each with their own unique colour. This one is coloured green, and so the green "h" key must be pressed prior to this key being pressed



The primary function of this key is coloured white

We could show this pictorially as follows: First when clearing individual digits click the backspace key each time a digit should be removed...



Second when clearing the entire X registers entry and returning to a zeroed display



### Functions using one number

One number functions only require one argument. Consider finding the logarithm (base 10) of the value 1.2149. We first type in the number 1.2149, and then execute the log command (which is the blue secondary function on digit 7). Pictorially, the key sequence would be...



The resulting display would show  $\text{Log}_{10} 1.2149$  as:-



#### Note



*In the keystroke diagram above we've shown the number 1.2149 as a set of six boxes each representing one keystroke. From now on we will abbreviate these keystroke diagrams to show numeric values using a single box – for example:-*



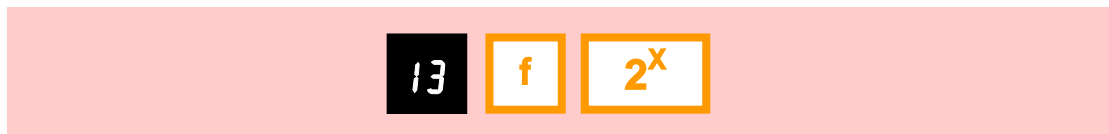
To find the trigonometric Sine of 45 degrees – then assuming the calculator is in degrees mode, type 45 and execute the SIN command (which is the yellow secondary function on the "B" key - top row). Pictorially, the key sequence would be...



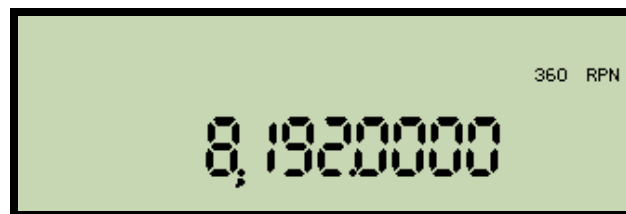
The resulting display would show Sin(45) as:-



A useful function for computer engineers is one that raises 2 to some value x. For example – if the address bus of a microcontroller has 13 bits connected to the memory IC's – how much memory can be addressed?



The display will confirm 8K of addressable memory can be accessed.



## Functions using two numbers

Two-number functions are functions that require two numbers in order to execute. Addition, subtraction, multiplication and division are all examples of two number functions (there are others, for example  $y^x$ ). These functions work the same way as one number functions – in the sense that the function itself (for example “multiply”) operates **WHEN** the key is pressed.

This means, that **BOTH** numbers must be present **WHEN** the key is pressed.

To place two numbers into the calculator and perform an operation

1. Key in the first number
2. Press the ENTER key to separate the first number from the second
3. Key in the second number
4. Press the desired function key.

All standard arithmetic operations are performed in this way.

















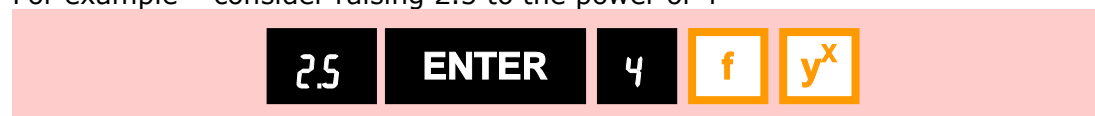
Solve	Key strokes	Display
$16 + 8$	   	24.0000
$16 - 8$	   	8.0000
$16 \times 8$	   	128.0000
$16 \div 8$	   	2.0000

Figure 4 – Table illustrating basic math two value operators

Another example of a two number function is  $y^x$ . This function is invoked when raising a number (Y) to a power (X). It works the same way as above ie:

1. Key in the first number (Y)
2. Press ENTER to separate the first number from the second
3. Key in the second number (X)
4. Execute the  $y^x$  function

For example – consider raising 2.5 to the power of 4



The display would show:-



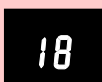















On the WP 34S there are actually two ways we can tackle this problem (and also any other similar one or two number functions involving reciprocals, roots and sigma data entry).

This particular shortcut makes use of a special set of keys at the top of the calculator labelled A, B, C and D. If these keys are currently not being used (as labels) in any program(s) written on the calculator then respectively these keys perform sigma data addition, reciprocal, powers and square root.

Let us assume for arguments sake that you haven't yet written any programs involving these keys. Performing the same function (raising 2.5 to the power 4) would involve the following key sequence



Try the following simple examples

Solve	Key strokes	Display
$18^3$	   	5832.0000
$12^2$	   	144.0000
$400^{-5}$	   	.200000
$3^{18}$	   	387420489.000



## Chain calculations

The value of the RPN system becomes apparent when dealing with chain calculations which works in a similar way to a paper based solution.

Example: Solve  $(12 + 14) * 5$

On paper you would solve the portion in brackets first, and then multiply by 5

$$\begin{array}{r} 26 \\ \hline (12 + 14) \times 5 \end{array}$$

26 multiplied by 5 is: 130.

You work through the problem in much the same way on the calculator. First obtain the intermediate result of  $12 + 14$

Keystrokes	LCD Display
12	12
ENTER	12.0000
14	14
+	26.0000

...and then solve the multiplication for the final answer. Note that you don't use the ENTER key to store the intermediate result of 26. It is automatically stored inside the calculator as soon as you type the next number.

Continuing...

5	5
X	130.0000

Pressing the function key (X) gives the final answer of 130.

This highlights some of the key advantages of RPN over most other calculators.

- You only ever have to work with one function at a time. In effect the logic forces you to cut problems down to a manageable size.
- Intermediate results appear as they are calculated
- Pressing a function key executes immediately – which means you see the result immediately
- There is no need to take special action with intermediate results because they are stored automatically
- You approach problems just as you would with pen and paper.

## Setting the display format

Internally – the calculator always works with numbers that have full 16 digits precision along with a three digit exponent. Regardless of the fact that the internals of the calculator always work with 16 digits, there are two formats that can be used to control the display of results to the user.

1. The number can be rounded to a specific number of digits. There are three modes that achieve this...
  - a. SCI (scientific notation) mode – fixes the number of decimal places and rounds the display to provide the desired number of decimal places but this format always uses a free ranging exponent. This mode is particularly useful when numbers are very large or very small.
  - b. ENG (engineering notation) mode – fixes the number of decimal places and again rounds the display to provide the desired number of decimal places. Just like scientific mode, this format always uses an exponent – but one where the value is always in multiples of 3. For example 24 kHz ( $24 \times 10^3$ ), or 15.6μS ( $15.6 \times 10^{-6}$ ).
  - c. FIX (fixed-decimal notation) mode – fixes the number of decimal places. The final result is rounded to provide the desired number of decimal places. Note that this is the default mode for many HP RPN calculators.

OR...

2. The calculator can be set to display all the digits in a number (note that trailing zeros will be blanked). This is known as the “ALL” display mode format.

### Note






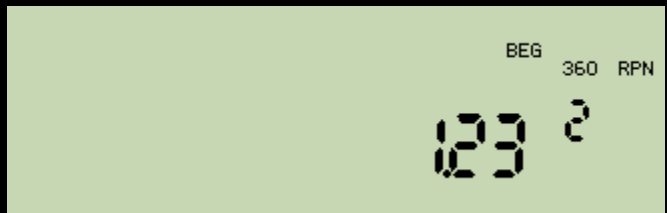








*Both the ALL and FIX modes of display have an interesting dilemma. What should the calculator do if the number in the X register has a large number of zeros following the decimal point? In other words at what point should the WP 34S forcibly switch to a better display mode? We will look at this issue further on*

## Scientific display notation - SCI

The scientific notation can be useful when working with large or small numbers. To select a scientific display rounded to 4 decimal places use the following sequence



Consider the following examples

















Keystrokes	Display
123.45678	Note that as soon as this value is entered, the calculator immediately converts the value to an exponent value of $1.2345678 \times 10^2$
  	
  	
  	

**Engineering display notation - ENG**

Engineering notation can be useful when working with large or small numbers in such a way that the exponent is always some multiple of 3. To select the engineering display rounded to 4 decimal places use the following sequence



Consider the following examples where we are displaying Pi divided by 9. Note that the exponent shows a power of 3.

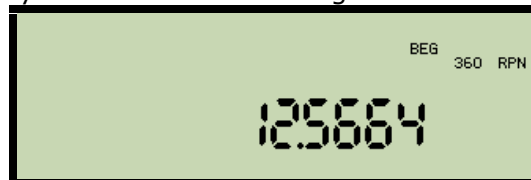
Keystrokes	Display
   	
  	
  	
  	

### Fixed display notation - FIX

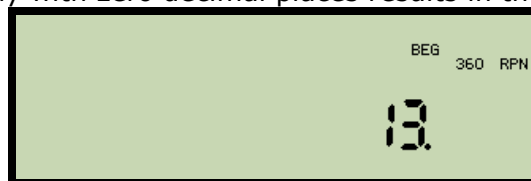
To use a fixed display showing 4 decimal places use the following sequence



With ( $\pi \times 4$ ) the display will show the following:-



Setting a fixed display with zero decimal places results in the following:-



So when the X register contains 12.5664 and the display mode is set to FIX but with zero decimal places then the calculator rounds up the display to 13.

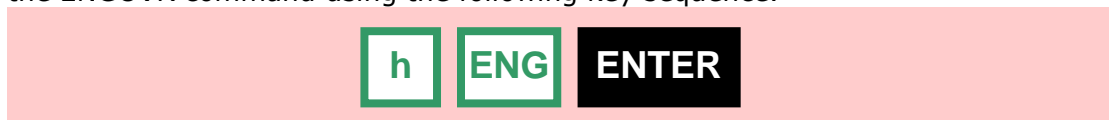
### Fixed display notation – Engineering / Scientific Override

As mentioned above, FIX has to cope with a specific problem occurring when the number of zero's after the decimal point exceeds the fixed number of places. For example consider fixing the display to four decimal places and then repeatedly dividing  $\pi$  by 10. On the fifth division, the X register would contain 0.000031416, and so the FIX command has to switch to a different mode (so that it doesn't misrepresent the number by displaying only 0.0000). By default it will switch to scientific mode, but you can override this default action and force the calculator to switch either to engineering or scientific mode. You do this using either the ENGOVR, or the SCIOVR command.

To set the calculator to switch into scientific mode, under these particular circumstances, invoke the SCIOVR command using the following key sequence.



Alternatively to switch into engineering mode under these circumstances invoke the ENGOVR command using the following key sequence.



With the value of Pi in the display divided 5 times by 10 - setting a scientific override results in a display of  $3.1416 \times 10^{-5}$ , whereas setting an engineering override results in a display  $31.416 \times 10^{-6}$

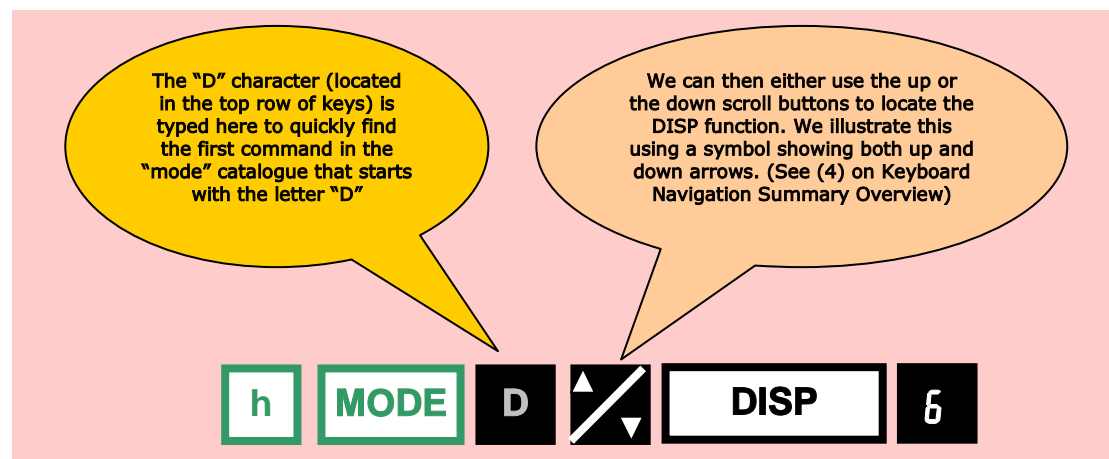
## ***Setting the number of display digits - DISP***

FIX, SCI and ENG each take an argument setting the number of digits displayed. For example FIX 4 will set the display to round with four digits after the decimal point (for example 3.1415).

There is an additional command in the MODE catalogue called DISP which allows the user to set the number of digits without changing the underlying display notation (FIX, SCI or ENG).

(When the display notation is set to ALL, then the DISP argument will change the switchover point – see ALL display mode notes below).




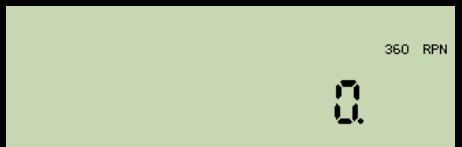




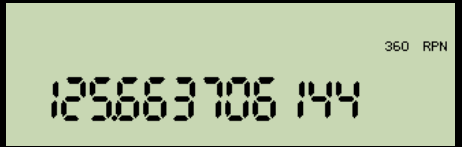








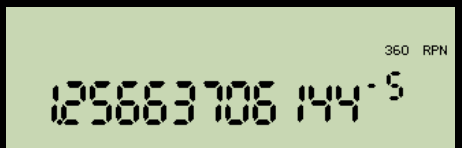
To switch to 6 decimal places while keeping the existing display notation – the following key sequence would be used.



## All display notation - ALL

As we've seen, the FIX, SCI and ENG display mode commands each take an argument defining how many characters should be displayed *after* the decimal point.

The ALL display mode is quite different. Regardless of the argument you provide to this command, setting ALL mode sets the calculator to display as many digits as it has available. The argument you provide is actually used to define how many zeros are allowed to exist after the decimal point before the calculator switches to a better display mode. Consider the following example where we start by taking Pi, multiplying by 40 and then dividing by 100000.

Keystrokes	Display
  	
   	
 	
Dividing this value again by 10 results in three zeros after the decimal point, and so as four zero's are required before a change occurs, the display remains as is.	
 	
Dividing by 10 one more time results in four zeros after the decimal point which is the threshold for switching. The calculator then switches to a different mode.	
 	

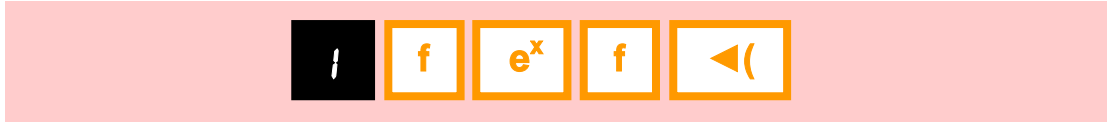
You'll notice that the calculator switches to SCI mode. Just as per the FIX display mode you can control which of engineering or scientific mode is used by invoking either the SCIOVR command to force scientific mode, or the ENGOVR command to force engineer mode (see the section above entitled "Fixed display notation – Engineering / Scientific Override")



### ***Viewing all digits in the Mantissa***

The calculator can be forced to reveal the first 16 digits of the mantissa and the exponent of a number using either of the horizontal scroll keys (◀ and ▶). Note that in this context both keys perform exactly the same function.

For example – enter  $e^1$  and click the left horizontal scroll to see the value of the constant  $e$  – using the following key strokes



The calculator will reveal the first 16 digits of the mantissa of the constant with the first four digits shown at the top left (2718), and the subsequent 12 digits shown on the line below (281828459045). The exponent is shown on the far right – and in this case an exponent of 000 indicates that the decimal point sits between the 1<sup>st</sup> and the 2<sup>nd</sup> digits (between the 2 and the 7).



The display confirms the approximation of  $e$  as: 2.718281828459045







## Setting the angular format

A number of trigonometric functions depend on knowing the particular angular format to use. These formats all describe an angle of rotation, but just adhere to a different standard. Generally most calculators ignore the rather uncommon format known as turns, and instead offer the three most popular formats namely degrees, radians and gradians. The WP 34S is no different.

Degrees range from  $0^\circ$  to  $360^\circ$ , radians from 0 to  $2\pi$  and gradians from  $0^g$  to  $400^g$ . Some of the most common angles in each of the three formats are compared in the table below

Units	Angular (rotation)							
Degrees	$0^\circ$	$30^\circ$	$45^\circ$	$60^\circ$	$90^\circ$	$180^\circ$	$270^\circ$	$360^\circ$
Radians	0	$\pi/6$	$\pi/4$	$\pi/3$	$\pi/2$	$\pi$	$3\pi/2$	$2\pi$
Gradians	$0^g$	$(100/3)^g$	$50^g$	$(200/3)^g$	$100^g$	$200^g$	$300^g$	$400^g$

To set the calculator into a desired angular format use the following key strokes

Action	Keystrokes
Sets the calculator to work in degree mode	 
Sets the calculator to work in radian mode	 
Sets the calculator to work in gradian mode	 






Note these are not conversions, and should not be thought of as such. Instead you are switching the calculator into the mode where (for example) if radians were set, then the calculator would treat  $2\pi$  as one full angular rotation. As such a subsequent call to take the Cosine of  $2\pi$  would return 1.0000.

See the Unit Conversions section if you wish to convert from one unit to another.

## Entering Exponents

Assume for the sake of this discussion that the calculator is now set to a fixed display rounded to four digits.

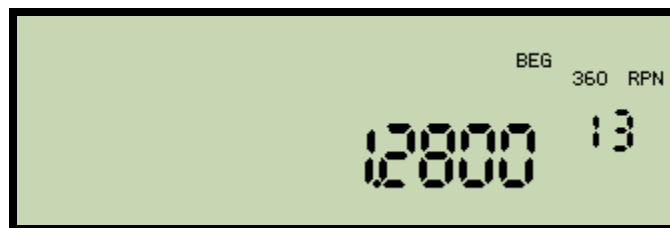
You can enter numbers multiples by powers of 10 by using the EEX (enter exponent key). For example – key in 12.8 trillion ( $12.8 \times 10^{12}$ ) and multiply it by 32

					
---	---	---	---	--	---

After the exponent is entered, and before the Enter is clicked the display will show what has been typed.



Once the "ENTER" key is clicked, the display will be corrected to show one integer place only (and the Exponent will be raised by a factor of 10).



The final multiplication by 32 gives the result



Note that directly **after** EEX is clicked, the "+/-" key can be pressed to change the sign of the exponent. So for example the following keystrokes would enter  $12.8 \times 10^{-12}$



Resulting in the following display



## Catalogues

The WP 34S contains a great many built in functions designed to solve a range of mathematical problems - everything from statistics and probability right through to the complex domain.

The WP 34S uses a set of catalogues, where each holds a group of related functions. For example there is a catalogue called PROB holding probability functions and a separate catalogue called STAT holding statistics functions.

You can easily recognize catalogue names, because each has an underline on the keyboard.

The WP 34S contains the following catalogues.

Catalogue	Description
<b>h</b> <u>CONV</u>	This catalogue holds the full suite of conversion functions – for example temperature conversions such as °F to °C.
<b>h</b> <u>MODE</u>	This catalogue holds functions that control the calculator itself. For example functions that define how the fraction mode of the calculator behaves, and what stack size should be used. Other functions control the geographical preferences and the math system to use when employing integer modes of operation (ie: 2's complement).
<b>h</b> <u>CONST</u>	This catalogue holds a fixed range of constant definitions. Constants such as Faraday's constant, Euler's e and $\pi$ are all stored here.
<b>h</b> <u>PROB</u>	This catalogue holds a set of functions linked to probability theory.
<b>h</b> <u>STAT</u>	This catalogue holds a set of functions linked to statistics.
<b>h</b> <u>SUMS</u>	This catalogue holds a set of functions linked to the summation of data.
<b>h</b> <u>MATRIX</u>	This catalogue holds a set of functions linked to matrix manipulation.
<b>h</b> <u>TEST</u>	This catalogue holds a set of functions used to conditionally test registers (generally used within programs)
<b>h</b> <u>P.FCN</u>	This catalogue holds a set of functions most commonly used in programs.
<b>h</b> <u>X.FCN</u>	This catalogue holds a set of functions generally grouped as extra functions
<b>h</b> <u>CAT</u>	This catalogue reveals the functions in main memory.
<b>CPX h</b> <u>X.FCN</u>	This catalogue (opened by prefixing the X.FCN catalogue name with the complex CPX button) holds a set of extra functions dedicated to the complex domain.
<b>CPX h</b> <u>MODE</u>	A small catalogue with advanced functions for advanced users only. Note that later revisions of the firmware will move these functions to the P.FCN catalogue

### Note



*When you get to the section covering the alpha register, you will discover an additional set of catalogues used to hold special characters (for example Greek letters, mathematical symbols and punctuation etc). These are known as alpha catalogues.*

The best way to become familiar with catalogues is simply to use them. We can do that by setting up calculator preferences along with the date and time.

## Using catalogues to set calculator preferences

When the calculator is first commissioned, it will (normally) be fitted with a crystal to drive the on-board real time clock. Assuming that is true then the time and date can be configured (if this assumption is not true then the following command sequences may hang the machine. On later revisions of the firmware, there is a function in the TEST catalogue called XTAL? and which returns true if a crystal is installed in the machine).

Let's now look at the command sequences required to configure the WP 34S for UK operation and which allow us to set the time and date. To do that we will

employ a number of special functions located inside two of the built in catalogues. We will also set the display control feature.

## Setting UK preferences

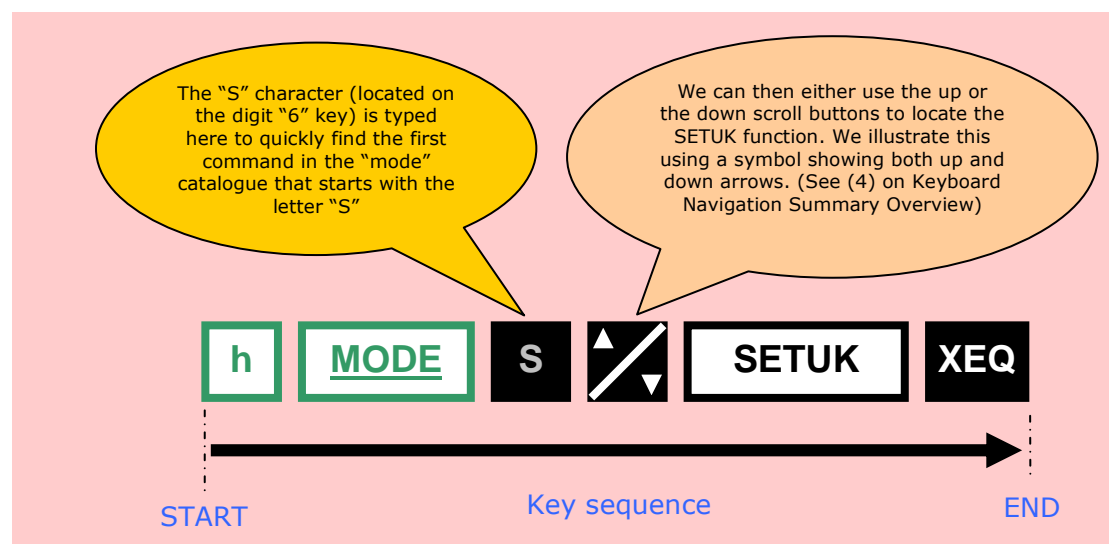
- (1) To set UK preferences (so that the calculator (a) uses a decimal point as the radix mark, (b) adds three digit separators, and (c) displays the time as 12 hour and the date in the format D.MMY) we would use the following key sequences
  - a. Click the green shift "h" key
  - b. Click the "MODE" function (on the CPX key – top right) which will open the mode catalogue and display the first function name in the list.
  - c. Click the "S" key (associated with the number 6 and shown on the bottom left of that key) which will prime the catalogue to quickly move to the first function in the alpha sorted list starting with the letter "S" (for example SEPOFF is one of the functions you will find)
  - d. As SETUK must be slightly further down in an alpha sorted list of functions you would repeatedly click the down arrow key until you located "SETUK" in the display.

### Note



*As you might expect, there is actually nothing to stop you scrolling the other way through the list of functions (using the up arrow key) but off course that approach does involve traversing virtually all functions in this alpha sorted list before finally reaching your desired function. The following notes assume that you've understood this point.*

- e. Click the XEQ function to execute the SETUK command OR click the EXIT button to cancel.



## Quickly locating catalogue functions

In the example above, you saw how you typed the letter "S" to quickly get the calculator to show all functions starting with the letter S. In fact this idea can be extended, in the sense that you can type two letters to further narrow the selection of functions on display. For example in the MODE catalogue if you typed the letter "S" followed by the letter "L" then the calculator would select the function "SLOW" because it is the only function starting with those two letters.

### Note



*In early revisions of the calculator firmware two letters used in this way had to be typed quite quickly. Later revisions of the firmware eliminate that requirement and make it altogether more relaxed. A timeout is still employed, but it is now very much longer (4-5 seconds or so). Later revisions also permit the backspace key to clear one or both typed letters.*

## Quickly locating catalogue functions with Greek letters

In addition – some functions in catalogues start with letters of the Greek alphabet. For example the constant catalogue "CONST" has definitions starting with the Greek letter gamma, and also the extra function catalogue X.FCN has functions starting with the Greek letter alpha (linked to the alpha register which we will look at later).

You can quickly locate these functions by pressing the Greek equivalent English letter prefixed with the blue (g) shift key. For example to quickly locate a function in the X.FCN catalogue starting with alpha, you would simply use the key sequence **g A** immediately after opening the catalogue.

## Fixing the display to four decimal places

It's a matter of personal preference how you would like the calculator to display floating point numbers. Some people prefer it to be fixed to four decimal places of precision (which was for example the default display mode for many of the older LED calculators such as the HP 32E). Some prefer it in scientific mode (where every number is shown with an exponent and which is very useful when dealing with very large or very small numbers), and some prefer engineering mode (where every number is shown with an exponent, but in multiples of 3). As an example – we will set our display to work with fixed four decimal places.

Later we will change this to six.

- (2) Set the calculator to fix the display to 4 decimal places
  - a. Click the green shift "h" key
  - b. Click the "FIX" function (under the "B" key on the top row)
  - c. Click the number 4 to set four digits of precision.

Again - we can show this sequence pictorially as follows



Note how the function FIX takes the number four as its value to use. We say FIX takes the argument 4.

## Setting the Time

To set the time you type the numeric time value into the X register using the 24 hour format H.MMSS and press enter. You then execute the "SETTIM" function which is located in the MODE catalogue.

The following examples will make the required time format clear

Time	X register contents
12:04:59am	00459
18:24:05	182405
1.27pm	132700
14:08:10	140810

Note the last example – if the calculator happened to be in a different display mode, then the trailing zero could become truncated. Keep that point in mind

- (3) To set the time in the calculator
  - a. Type the current time value into the X register (see above format) and press ENTER
  - b. Then find and execute the SETTIM function in the mode catalogue



## Setting the Date

To set the calculator date you first type the numeric time value into the X register using the format D.MMYYYY and then press enter. You then execute the "SETDAT" function which again is located in the MODE catalogue.

Remember earlier, we fixed our display to four decimal places – which means in the case of a date formatted with a value D.MMYYYY we would actually lose the two last digits of the year. So in order to avoid confusion, let's now fix to 6 decimal places.

- (4) Set the calculator to fix the display to 6 decimal places



The following date value examples will make the required date format clear

Date	X register contents
1 Sept 2018	1092018
24 July 2019	24072019
1 Jan 2020	1012020

- (5) To set the date in the calculator
  - a. Type the current date value into the X register (see above format) and click ENTER



- b. Then find and execute the SETDAT (set date) function in the mode catalogue



### Reading the Time from the calculator

Leaving the calculator set to fix the display to 6 decimal places, you can read the time and date using the "TIME" and "DATE" functions in the extra function (X.FCN) catalogue.

The following sequence will display the current time

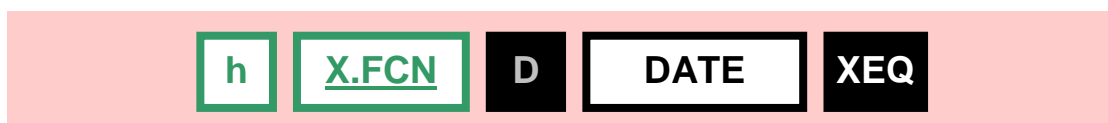


The resulting display at 15:33 (and 45 seconds) would be...

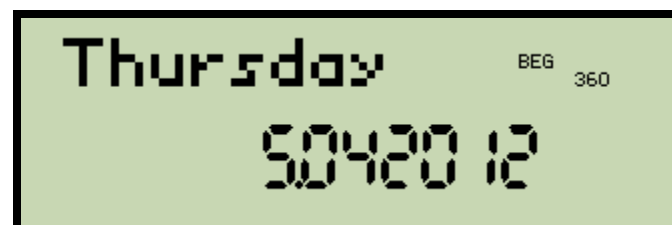


### Reading the Date from the calculator

The following command sequence will display the current date



The resulting display on Thursday the 5<sup>th</sup> April 2012 would be...



Observe that if you had fixed the display to 4 digits, the year would be truncated.

## The Automatic Memory Stack

### Note



*The following notes are based on the description used in the manual supplied and written by Hewlett-Packard and entitled "Solving problems with your Hewlett-Packard calculator" supplied way back in 1978 when a still working and much loved LED HP 32E calculator was originally purchased.*

*A great many HP calculators make use of a four register stack (using names X, Y, Z and T). However, the WP 34s has a configuration option in the mode catalogue (see SSIZE8 and its complement SSIZE4) which allows the stack to be doubled in size (X, Y, Z, T, A, B, C and D). The two stack sizes work identically except there is more room for intermediate results with an 8 level stack. Assume for the sake of this discussion that the WP 34S is configured to use a four level register stack which in practice is usually more than adequate for even very involved tasks.*

The automatic storage of intermediate results is the key to understanding how reverse polish notation works. The heart of this calculator is its stack. Master the stack, and you will master the machine.

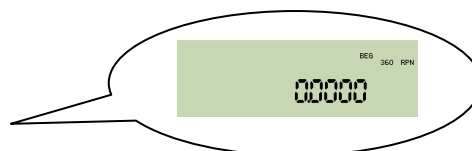
### Initial Display

Numbers are stored and worked on in machine registers. The displayed X register (which is the only easily visible register) is actually one of four registers which collectively make up the stack. These registers are known as X, Y, Z and T. Any number (no matter what size) can be contained in any one of these four registers.

Registers are stacked one on top of the other, with the displayed X register at the bottom.

### Register Contents

T (top)	0.0000
Z	0.0000
Y	0.0000
X	0.0000



The X register is always displayed

## Stack control - manipulation

There are a few key functions that you can use to control where things are on the stack. For example the roll up, and roll down keys, and also the X exchange Y key. Let us now look at each in turn.



Primary function for this key is to roll the stack down

Note that a secondary function (green shift) is to roll the stack up.

Primary function for this key is to exchange the contents of the X and Y registers on the stack



## Roll Down (and Roll Up) Key

To see how the roll down key works, load the stack with the numbers 1 through 4 by pressing.



The numbers will now be loaded into the stack and its contents will look like this

Register	Contents
T (top)	40000
Z	30000
Y	20000
X	1

Each time the roll down key is pressed, the stack contents will shift downwards one register. By pressing roll down, the last number keyed in (in this case 1) will be rotated into the T register.

Pressing **R↓** for the first time will result in the following (note how the X register is written into the T register)

<b>Register</b>	<b>Contents</b>
T (top)	10000
Z	40000
Y	30000
X	20000

Pressing **R↓** for a second time will result in the following

<b>Register</b>	<b>Contents</b>
T (top)	20000
Z	10000
Y	40000
X	30000

Pressing **R↓** for a third time will result in the following

<b>Register</b>	<b>Contents</b>
T (top)	30000
Z	20000
Y	10000
X	40000

And pressing **R↓** for a fourth time will return to the starting point

<b>Register</b>	<b>Contents</b>
T (top)	40000
Z	30000
Y	20000
X	10000

Roll up (**R↑**) works the same way, but just in reverse (note it is a secondary function, and shown with a green colour – so you must prefix it by pressing the green “h” key). You can use roll to review the contents of the stack. Just press each roll (up or down) respectively 4 times to return the contents to their original registers.


#### Note



*Don't use Roll immediately after using ENTER, or CLx.*


*You'll understand the reason why shortly.*

## Exchanging X and Y

The  key simply swaps the contents of the X and the Y register.

With the data in the previous example, clicking the X exchange Y key will change the numbers in the X and Y registers to

<b>Register</b>	<b>Contents</b>
T (top)	4.0000
Z	3.0000
Y	1.0000
X	2.0000

Pressing  again returns the stack to its original positions.

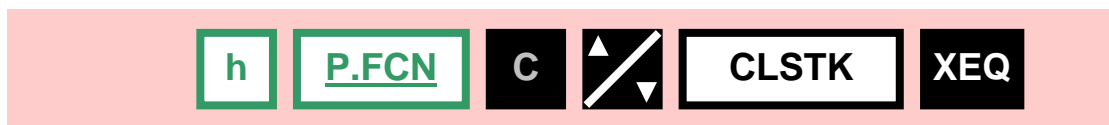
<b>Register</b>	<b>Contents</b>
T (top)	4.0000
Z	3.0000
Y	2.0000
X	1.0000

## Clearing the Stack

If you wish to clear the entire stack – you can do it in one of two ways. You could for example manually overwrite each stack register with zero as follows



Or you could open the P.FCN (programming function catalogue) and use the CLSTK (clear stack) command. When the catalogue is open press the "C" key to quickly jump to all commands starting with the letter "C", and then use either the scroll up, or scroll down keys to locate the CLSTK command.



## The ENTER Key

Assume that the stack is now cleared to zero in every register

When a number is typed into the calculator, the contents are placed into the displayed X register. Key in the value 178.4217, and the stack registers will look as follows

Register	Contents
T (top)	0.0000
Z	0.0000
Y	0.0000
X	178.4217

To key in a second number, you must first separate the digits of the first number (178.4217) from the digits of the second. One way to separate numbers is to press the ENTER key.

Press the **ENTER↑** key now and the register contents will change as follows:-

Register	Contents
T (top)	0.0000
Z	0.0000
Y	178.4217
X	178.4217

Pressing ENTER causes the numbers in the displayed X register to be copied into the Y register. Similarly, numbers in the Y and Z registers have both been copied into the Z and T registers and whatever number was stored in T has been lost off the top of the stack. This chain of events is known as stack "lift"

After pressing ENTER the X register is then effectively prepared for a new number – and whatever is typed by the user simply overwrites the current contents of the displayed X register.

For example – now key in the value 256.17. The contents of the registers will change to...

<b>Register</b>	<b>Contents</b>
T (top)	0.0000
Z	0.0000
Y	178.42 17
X	256.17

Note that at this point, the CLx key (**CLx**) replaces any typed numbers with zero – which means we can remove errors. Imagine we meant to type in 128.17 instead of 256.17. Then click CLx, and the registers will change to

<b>Register</b>	<b>Contents</b>
T (top)	0.0000
Z	0.0000
Y	178.42 17
X	0.0000

Now type 128.17 and the registers become

<b>Register</b>	<b>Contents</b>
T (top)	0.0000
Z	0.0000
Y	178.42 17
X	128.17

#### Note



*Numbers already in the stack don't move when a new number is typed in directly after clicking ENTER or CLx. However, numbers in the stack DO move up when a new number is typed in directly after the Roll key is used. This is why we earlier warned not to review the contents of the stack with roll directly after pressing ENTER or CLx*

## Stack Arithmetic

Assume that the stack is now fully cleared to zero.

The calculator performs arithmetic in much the same way you would if you were using pencil and paper. Imagine you wished to add the two numbers 24 and 31. You would use the following sequence to enter the numbers



The stack would now look as follows

Register	Contents
T (top)	000000
Z	000000
Y	24.00000
X	31.

With the two values positioned on the stack vertically, we can then add by pressing the Add (+) button.

### Note



*This is a general rule with no exceptions. You first position both numbers in the stack. You then execute the operation (for example add, subtract, multiply, divide) by pressing the function key.*

So for example to add 24 and 31

Keystrokes	Display
24	24.
ENTER	24.00000
31	31.
+	55.00000

Subtract 31 from 24

Keystrokes	Display
24	24.
ENTER	24.00000
31	31.
-	-7.00000

Multiply 24 by 31

Keystrokes	Display
24	24.
ENTER	24.00000
31	31.
X	744.00000

Divide 24 by 31



Keystrokes	Display
24	24.
ENTER	24.0000
31	31.
÷	0.7742

Raise 24 to the power of 31

Keystrokes	Display
24	24.
ENTER	24.0000
31	31.
$y^x$	6.1171 <sup>42</sup>

The result of raising Y to the power X above is:  $6.1171 \times 10^{42}$

Note that the  $y^x$  function is available as a secondary function on the 9 key (using the yellow **f** shift), or if no programs have been created that use the label "C" it will be assigned as a primary function on the top row "C" key

#### Note



When raising  $y^x$  it can help to remember that Y and X register names map one-to-one to this function.

Given you will enter two values – the first entered will end up in the Y register due to stack lift. The second value will be in the X register when you invoke the  $y^x$  function

## Chain Arithmetic

In each calculation performed above, numbers were manually positioned in the stack using the ENTER key. The stack itself is organised so that positioning is done automatically – specifically so that intermediate results are in an ideal position.

Every number that results from a calculation in the stack is "lifted" automatically when a new number is keyed into the calculator. The stack "knows" that after it completes a calculation, any new digits keyed in are part of a new number.

The stack also does the reverse. It "drops" automatically when you perform an operation (for example ADD takes two values in X and Y, and leaves one result and so the stack drops).

To see this in action consider the calculation  $(16+30+11+17)$ . We will tackle this by placing all four values on the stack at the start and then execute each of the three add operations. The full sequence would be:-

16	ENTER	30	ENTER	11	ENTER	17	+	+	+
----	-------	----	-------	----	-------	----	---	---	---

Let us now look at how each of these steps alters the stack

16	<table> <tr> <th colspan="2">Stack Contents</th></tr> <tr> <td>T</td><td>000000</td></tr> <tr> <td>Z</td><td>000000</td></tr> <tr> <td>Y</td><td>000000</td></tr> <tr> <td>X (display)</td><td>16.</td></tr> </table> <p>16 is typed into the display register</p>	Stack Contents		T	000000	Z	000000	Y	000000	X (display)	16.
Stack Contents											
T	000000										
Z	000000										
Y	000000										
X (display)	16.										
ENTER	<table> <tr> <th colspan="2">Stack Contents</th></tr> <tr> <td>T</td><td>000000</td></tr> <tr> <td>Z</td><td>000000</td></tr> <tr> <td>Y</td><td>160000</td></tr> <tr> <td>X (display)</td><td>16.0000</td></tr> </table> <p>16 is copied into Y</p>	Stack Contents		T	000000	Z	000000	Y	160000	X (display)	16.0000
Stack Contents											
T	000000										
Z	000000										
Y	160000										
X (display)	16.0000										
30	<table> <tr> <th colspan="2">Stack Contents</th></tr> <tr> <td>T</td><td>000000</td></tr> <tr> <td>Z</td><td>000000</td></tr> <tr> <td>Y</td><td>160000</td></tr> <tr> <td>X (display)</td><td>30.</td></tr> </table> <p>30 is typed into the display register. It overwrites the 16</p>	Stack Contents		T	000000	Z	000000	Y	160000	X (display)	30.
Stack Contents											
T	000000										
Z	000000										
Y	160000										
X (display)	30.										
ENTER	<table> <tr> <th colspan="2">Stack Contents</th></tr> <tr> <td>T</td><td>000000</td></tr> <tr> <td>Z</td><td>160000</td></tr> <tr> <td>Y</td><td>300000</td></tr> <tr> <td>X (display)</td><td>30.0000</td></tr> </table> <p>30 is copied into Y, and the stack is lifted which means that the value that was in Y is now moved up to Z. Similarly the value in Z is moved to T</p>	Stack Contents		T	000000	Z	160000	Y	300000	X (display)	30.0000
Stack Contents											
T	000000										
Z	160000										
Y	300000										
X (display)	30.0000										
11	<table> <tr> <th colspan="2">Stack Contents</th></tr> <tr> <td>T</td><td>000000</td></tr> <tr> <td>Z</td><td>160000</td></tr> <tr> <td>Y</td><td>300000</td></tr> <tr> <td>X (display)</td><td>11.</td></tr> </table> <p>11 is typed into the display register. It overwrites X</p>	Stack Contents		T	000000	Z	160000	Y	300000	X (display)	11.
Stack Contents											
T	000000										
Z	160000										
Y	300000										
X (display)	11.										
ENTER	<table> <tr> <th colspan="2">Stack Contents</th></tr> <tr> <td>T</td><td>160000</td></tr> <tr> <td>Z</td><td>300000</td></tr> <tr> <td>Y</td><td>110000</td></tr> <tr> <td>X (display)</td><td>11.0000</td></tr> </table> <p>11 is copied into the Y register, and again the stack is lifted. The old contents of Y moves to Z. The old contents of Z moves to T. The old contents of T is lost.</p>	Stack Contents		T	160000	Z	300000	Y	110000	X (display)	11.0000
Stack Contents											
T	160000										
Z	300000										
Y	110000										
X (display)	11.0000										

Cont...

...Cont

17	<b>Stack Contents</b>	
	T	16.0000
	Z	30.0000
	Y	11.0000
	X (display)	17.
17 is types into the display register. It overwrites X.		
+	<b>Stack Contents</b>	
	T	16.0000
	Z	16.0000
	Y	30.0000
	X (display)	28.0000
Add is a two value function. So, when + is clicked, the values of 17 and 11 are added together and the stack is dropped one time. 16 Drops from T into Z (nothing drops into T and so T holds its original value – a feature that is useful see constant arithmetic). Note that the values 30 and 28 are now ready for whatever the next operation might be (in this case add)		
+	<b>Stack Contents</b>	
	T	16.0000
	Z	16.0000
	Y	16.0000
	X (display)	58.0000
30 and 28 are added together. The stack drops again. Now 16 and 58 are ready for the next operation.		
+	<b>Stack Contents</b>	
	T	16.0000
	Z	16.0000
	Y	16.0000
	X (display)	74.0000
16 and 58 are now added together for the final answer. The stack continues to drop.		

Exactly the same dropping action occurs with any command such as subtraction, multiplication and division. Any two-value-function will take inputs from the X and Y register and the stack will be dropped as they are taken. The function will then perform its particular operation and leaves its result in the visible X register.

This automatic stack lift and stack drop affords great advantages because intermediate results are easily retained and positioned in just the right place during long calculations.

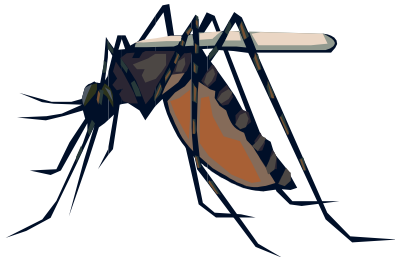
You may have noticed that as the stack was dropped and everything moved down, the T register (at the top of the stack) didn't change. The top of the stack isn't altered by a stack drop.

This is a feature which can be used for constant arithmetic

## Constant Arithmetic

Whenever the stack drops (ie: when a two value operation occurs) the number in the T register drops into the Z register, but note that the T register remains unchanged. In the above example, after each stack drop the T register continued to hold the value 16.

Consider the following example where we can use this feature.



There is a mosquito whose population increases by 17.2% each day under certain conditions.

If a naturalist starts a sample population of 1000, what will be the population at the end of each day over six consecutive days?

Keystrokes	Display
1.172	1.172
ENTER	1.1720
ENTER	1.1720
ENTER	1.1720
1000	1000.
X	1172.0000
X	1373.5840
X	1609.8404
X	1886.7330
X	2211.2511
X	2591.5863

When X is clicked the first time, the calculator computes  $1.172 \times 1000$ . The result 1172.0000 is displayed in the X register and a new copy of the growth factor drops into the Y register. Since a new copy of the growth factor is duplicated into the T register each time the stack drops, there is no need to manually key in the value.

Note that performing two-value functions such as multiply causes the number in the T register to be duplicated each time the stack drops.

By contrast the roll keys (explained above) cannot perform this function. These functions rotate the contents of the stack registers, but do not rewrite any number. They merely shift numbers that are already in the stack.

## Order of Execution

When solving a problem like this one

$$5 \times \frac{((3/4) - (5/2) + (4 \times 3))}{(3 \times .213)}$$

You must first decide where to start before pressing any keys. The rule of thumb is always to start at the innermost set of parenthesis and work outward. In this case, an experienced RPN eye will probably attempt to clear the top line first, then the bottom line of the right hand side fraction and then finally multiply the lot by 5 either with a 5 that was entered on the stack right at the start, or at the end. Personally I would write the 5 at the start – but that's just me.

Keystrokes	Display	Comment
5	5.	
ENTER	5.0000	Load the left hand side multiplier onto the stack. We will hold this on the stack for use right at the end of the calculation. It nicely demonstrates why the stack is such an asset in long calculations.
3	3.	
ENTER	3.0000	
4	4.	
/	0.75	This value is our first intermediate result
5	5.	
ENTER	5.0000	
2	2.	
/	2.5000	We can now subtract this value from our previous intermediate result leaving -1.75
-	- 1.7500	This is our next intermediate result
4	4.	
ENTER	4.0000	
3	3.	
X	12.0000	We can now add this value to our previous intermediate result leaving 10.25 – which clears our top line and which we now hold as an intermediate result
+	10.2500	
3	3.	
ENTER	3.0000	
.213	0.213	
X	0.6390	We can now divide this into the previous intermediate result for the top line
÷	16.0407	
X	80.2034	Remember we loaded 5 onto the stack right at the start. We can now multiply the value of the right hand side of the equation with that value to form the final answer of 80.2034

## LAST X

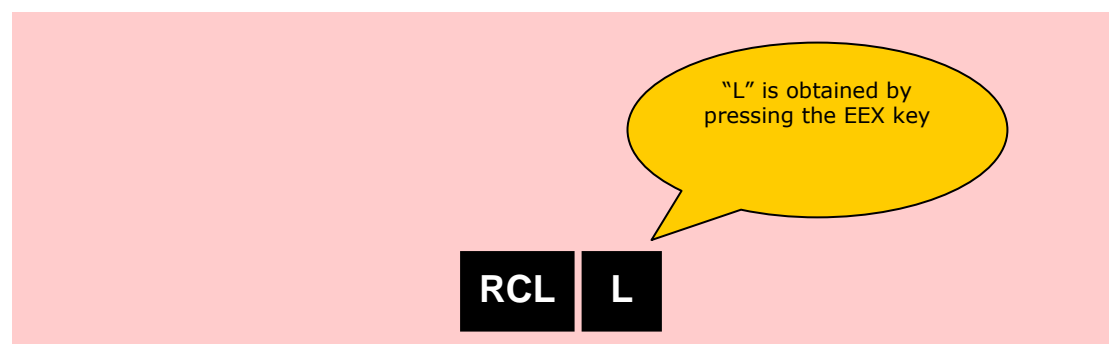
In addition to the four stack registers that automatically store intermediate results, the WP 34S calculator also contains a useful and separate automatic register called the last X register. This register preserves the value that was last displayed in the X register before the performance of a function.

### Note



*In days of old, HP calculators had a dedicated LST X key on the keyboard. You simply pressed that key to place the contents of the last X register into the X register. Alas, you won't find that key on a WP 34S but you will find something nearly as good...*

The WP 34S maps the last X register to register "L". You can therefore type RCL "L" (where the letter L is a secondary function of key EEX) to place the last X value back into the X register.



RCL "L" is exactly the same as executing "LST X" on other HP calculators. The last X feature makes it easy to recover from keystroke mistakes such as pressing Add when you meant multiply.

For example: Divide 15 by 2.199 after you have divided in error by 3.199

Keystrokes	Display	Comment
15	15.	
ENTER	15.0000	
3.199	3.199	Accidentally type in 3.199 when in fact you meant to type in 2.199
/		Oops – we've now divided with the wrong value.  So we need to cancel this out. First recover the LAST X value – which is in register L on the WP 34S. Note that the "L" ASCII character is located on the "EEX" key
RCL "L"	3.199	As we divided originally, do the opposite to cancel
X	15.0000	
2.199	2.199	Enter the correct value
/	6.8213	and arrive at the final result

Another time when LAST X can be useful is when a single value appears in a calculation more than once. Consider the following calculation:-

$$\frac{8.1142 + 9.18368176}{9.18368176}$$

Keystrokes	Display	Comment
8.1142	8.1142	
ENTER	8.1142	
9.18368176	9.18368176	
+	17.2979	
RCL "L"	9.1837	Use the LAST X function to retrieve the long number for a second time.
/	1.8835	

### **Complex domain LAST X**

When working in the complex domain the WP 34S has a last X equivalent. The calculator employs two registers (I for the imaginary part, and L for the real part).

Obtaining the complex last X involves the following key strokes



The screen display will confirm that a complex result is being shown by placing a C in the top left corner.

## Memory Organisation

The WP 34S contains 2KB of non volatile RAM (*memory that retains data when the device is switched off*) organised as four distinct sectors.

Starting from the very top of memory and working down we have...

1. Status and configuration data – a small immovable block typically 50 to 100 bytes containing status and calculator mode data along with the contents of the alpha register and a set of 14 bytes used to hold 112 (ie: 14x8) global user flags. The user has no direct access to this memory, and must use assorted calculator commands to read/write to this area.

### Note



*The status and configuration area of memory is subject to change based on features offered in future revisions of the calculator firmware.*

2. Global registers (ie: holding the general purpose registers accessed for example by STO and RCL, and the RPN stack (X,Y,Z and T for the four level stack, and the additional A, B, C and D registers when the device is configured with an 8 level stack).
3. Registers used for cumulative statistics – namely the sigma registers. Observe that these are quite separate from the normal general purpose registers (unlike many HP calculators where these two uses are shared). Splitting these registers into a separate block provides safeguards in the form of encapsulation and also assist precision.
4. The subroutine return address stack (SBR) used to hold subroutine return addresses (note this stack has nothing to do with the normal 4 or 8 level RPN stack and merely deals with the program addresses pushed and popped off the stack during normal program execution calls and returns).

### Note



*A complete copy of non-volatile RAM can be written to the semi-permanent flash memory using the SAVE (or the ON+STO) commands*

Memory can be traded between program steps, general purpose registers and the flags but it is important to realise that the calculator will never provide a full complement of program steps AND registers AND flags all at the same time.

One must therefore trade, based on the intended application.



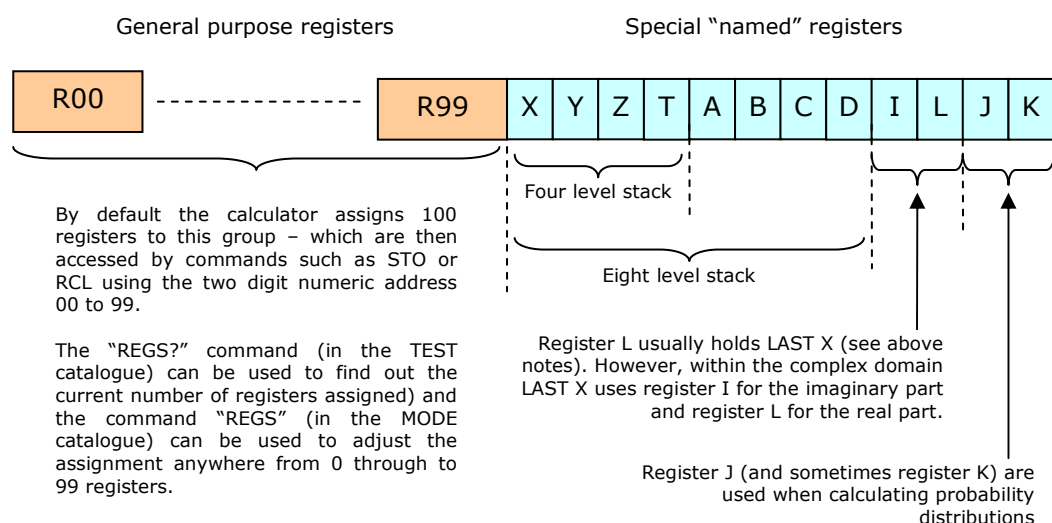
## Global registers

Global registers are placed close to the top of memory and split into two parts.

The first part holds the registers that can be accessed via the normal STO and RCL commands familiar to HP users of old. By default a total of 100 registers are allocated by the calculator for this block and the user would use 00 to 99 to address each individual register.

The second part of this area is fixed, and holds the special named registers used to manage the RPN stack and which are individually named as X, Y, Z, T, A, B, C, D, I, L, J and K. Keep in mind that you generally use the alpha name to access each of these registers.

The two blocks combine (by default) to give a total of 112 registers as follows:-



The RPN stack (ie: the 12 named registers shown in light blue above) are fixed and cannot be altered. The remaining register numbers are configurable. By default 100 registers are assigned (00 to 99) however the "REGS" command located in the MODE catalogue can be used to reduce the assigned number anywhere from 99 right down to zero if no registers were required.

The "REGS?" command in the TEST catalogue can be used to determine the current assignment.

## Global Register Access – Direct and Indirect

An important concept to understand is that the assigned block of general registers described above (100 registers by default addressed as 00 to 99) can be addressed either directly or indirectly.

A worked example will help make this clear.

First we will use direct addressing to write the value 24 into register 31

24 STO 31

We will then read the register 31 back after first clearing the X register

h CLx RCL 31

At which point the display will show



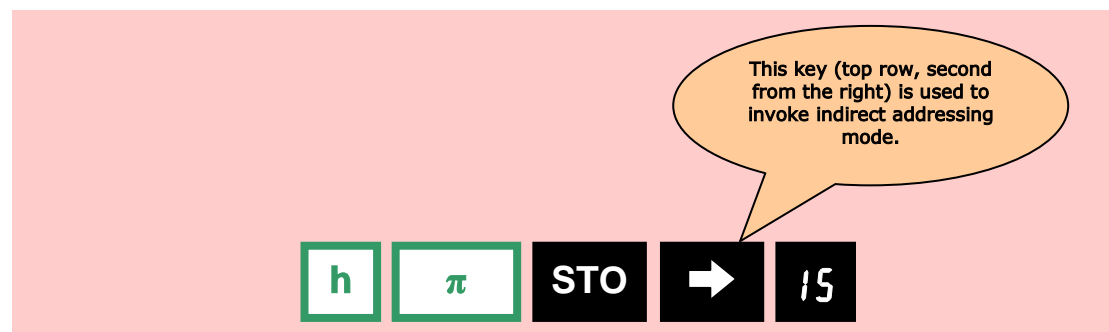
Next we will use indirect addressing.

In this mode we involve two registers, where the first is used to hold the address of the 2<sup>nd</sup>. We will use register 15 to hold the address of register 2, and we will then indirectly update register 2.

First prime register 15 so that it holds the target register number (in this case 2)

2 STO 15

Next, indirectly write the value of PI to register 2 via register 15



If you now recall register 2, you will find the value 3.1415 stored, whereas register 15 will still contain the unchanged value 2.

Recall register 2 (RCL 02)	Recall register 15 (RCL 15)

Indirect addressing is a very powerful mechanism that can be used to implement smart lookup tables and dynamic data arrays. It is also sometimes the only way certain registers in the WP 34S can be accessed when the register number exceeds 99.

Indirect addressing has generally all sorts of interesting applications.

## Global Register Access – Out Of Range Error

Keeping in mind that the number of assigned registers can be changed via the REGS command (in the MODE catalogue) then assume for arguments sake that the REGS command has been used to assign a total of 32 general purpose registers (00 to 31 instead of the default 00 to 99).

In this case the user will be able to either directly AND/OR indirectly access up to register 31 and no further.

The keyboard user interface will actively prevent any attempt to access registers 32 to 99, and any program line attempting the same thing will raise an Out Of Range exception error.

## Cumulative Summation Registers – Sigma Data

The memory needed for cumulative statistics is allocated separately from the general registers. This separation safeguards one data set from the other (and is helpful in terms of the precision of the device).

The ONLY way to update the summation registers is via the  $\Sigma+$  and  $\Sigma-$  keys. The data stored in this area of memory is then evaluated and accessed by specific functions and only those functions. This data is not available via STO or RCL.

The first time  $\Sigma+$  is used, a block of 14 summation registers are created (70 words) in between the general purpose (and stack) registers and the subroutine return stack.

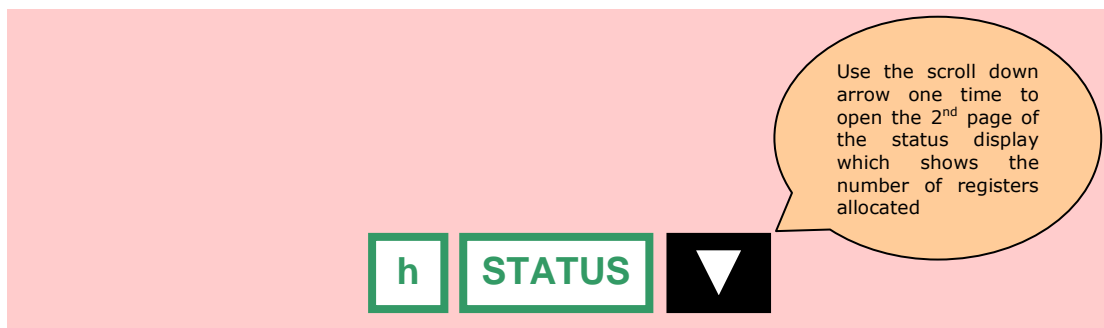
**Note**



*As you can imagine, it is possible that a configuration of memory previously set in the calculator could prevent this allocation of sigma data registers being made until room is made first.*

After the  $CL\Sigma$ ,  $CLALL$  or  $RESET$  functions are invoked, the memory allocated to the summation registers will be released (a process that occurs entirely transparently to the user and programs running on the calculator).

On the calculator (not the computer simulation) you can tell as/when the allocation of sigma data registers has been made by viewing the **STATUS** command. Before using the  $\Sigma+$  sigma function use the following command sequence.



With the default allocation of 100 general purpose registers the display will show



Press the **EXIT** key to cancel the **STATUS** display.

Next, use the following key strokes to enter an X, Y value (3, 8) into Sigma



And then reopen the STATUS command and select the 2<sup>nd</sup> page.



With the default allocation of 100 general purpose registers, and our first sigma point input to the calculator, the display will now show a  $\Sigma+$  symbol next to the "Regs" word in the display.

(Note that this does not work on the WP 34S computer simulator)

Press the EXIT key to cancel the STATUS display.

## ***Program Steps & Subroutine Return Stack***

The maximum number of program steps is 926, ranging from step 000 to step 925.

The subroutine return stack is used to hold return addresses that are pushed onto the stack when a function is called, and popped off the stack when the function executes the return (RTN) command. There is no command to set the size of this area, as it simply grows down from the top most program step currently stored.

The subroutine return stack is also used to hold local variables and flags described next.

## **Local Data**

A mechanism exists for creating a block of data from the overall memory pool that is local to a program. Please refer to the local data (variables and flags topics) in the programming section of this guide for more details.

## Clearing memory to make space

If you need more memory space – for example if you run out of program memory then you have the following options.

1. You can reduce the number of global registers (normally 100 are assigned) using the REGS command. Each register is roughly equivalent to four program steps
2. You can move programs to flash memory and then clear the respective sections in RAM. A set of four program steps is roughly equivalent to a single register.
3. You can release the sigma registers (assuming they have been allocated) by invoking the CLΣ command.

Use the STATUS command to monitor progress.

## Backing up the calculator memory - introduction

The WP 34S provides a mechanism to create and retrieve a backup of all calculator memory including program storage, the numbered registers, the sigma registers and calculator state.

The calculator is capable of storing one backup in flash ram. When the machine is first commissioned, a new backup is created in this area that is cleared of all data. A point to note here is that if the calculator is ever hardware reset (see section at the end of this guide) the contents of the flash RAM backup will automatically be restored into the calculator main memory. So - if the backup is blank, then the calculator will be cleared after the reset.













### Note























*The backup will remain blank until after you create your first backup using some combination of the commands shown below. Therefore you should get into the habit of making regular calculator backups as you add programs and make changes – all the more so given it is extremely easy to do.*

Managing backups requires the use of a small collection of commands in the programming function catalogue (**P.FCN**) – which can be summarised as first allowing the user to save everything and second retrieving part or all of the previously saved data.

The following table shows the commands available in the **P.FCN** catalogue which include SAVE, LOAD, LOADP, LOADR, LOADSS and the LOADΣ command.

Command	Description
     	Creates a full backup of main memory into flash RAM. This command saves all user program space, all registers and the current settings and state of the calculator. The command will overwrite any previously created backup. The display will show "Saved" when the command is executed.
     	Restores a previously created backup of main memory from flash RAM. This command overwrites the existing program space and all registers and the current settings and state of the calculator with whatever was in the backup data. LOAD is functionally equivalent to executing LOADP, LOADR, LOADSS and LOADΣ. The display will show "Restored" when the command is executed.

    	Restores the complete program memory from a previously created backup in flash RAM and <b>appends</b> that to the existing programs in RAM (note that this will only work if there is enough program memory space to cope)
    	Restores numbered general purpose registers from a previously created backup in flash RAM. Note that lettered registers (X,Y,Z,T,A,B,C,D,I,L,J and K) are not restored. The number of registers copied is the minimum of the registers held in the backup and RAM at execution time.
    	Restores the system state from the previously created backup in flash RAM.
    	Restores the summation registers from a previously created backup in flash RAM. Note an exception will be raised if no backup exists.

### ***Backing up the calculator memory – hot key shortcut***

You can use SAVE and LOAD as shown above. However, the WP 34S provides an elegant shortcut key combination which can be used to directly invoke either the SAVE or the LOAD without having to navigate a catalogue.

### **Creating a new backup using ON & STO**

To create a new backup (which will overwrite the existing backup) simply following the exact sequence shown below

1. Press and hold the ON key (ie: the EXIT key) – keep this held down
2. Press and release STO (keep the ON key pressed)

The display will then display an “Are you sure” style challenge with the words...

Backup?  
to FLASH

3. Press and release STO if you wish to proceed with creating the backup, or simply release the ON key to cancel.
4. Release the ON key

### **Restoring a backup using ON & RCL**

To restore a previously created backup follow the exact sequence below

1. Press and hold the ON key (ie: the EXIT key) – keep this held down
2. Press and release RCL (keep the ON key pressed)

The display will then display an “Are you sure” style challenge with the words...

Restore?  
FLASH

3. Press and release RCL a second time if you wish to proceed with the restoration, or simply release the ON key to cancel.
4. Release the ON key

## How **STO (+,-,/,x)** and **RCL (+,-,/,x)** access items in memory

With the above discussion in mind you might be wondering how one might store (STO) or recall (RCL) the value of a specially named RPN stack register (for example any of the 12 specially named registers: X, Y, Z, T, A, B, C, D, I, L, J and K). Generally if you execute a command such as STO or RCL the calculator will pause displaying the command name with an underscore prompt. What you type next controls precisely which command is invoked, and what argument is used as the input. For the notes below assume the following...

### Assumptions List

1. Register 2 contains 14, Register 5 contains 4, Register 14 contains 8
2. 16 Local data registers have been allocated by the LocR command (see above notes)
3. X register contains 3.

Let us first consider using the RCLx function (which recalls register n and then multiplies that value by the existing X register – leaving the product in X). Remember as we've said earlier, functions assigned to the keyboard generally never appear in any catalogue, so to invoke the RCLx command we actually need to press RCL immediately followed by the multiply key (the same idea applies to RCL+, RCL-, RCL/ and all the similar STO functions).

Function	1 <sup>st</sup> Key	2 <sup>nd</sup> Key	3 <sup>rd</sup> Key	Comment
RCLx	2	ENTER		Recalls the value of register 2 giving (3x14)=42 in X.
RCLx	1	4		Recalls the value of register 14 giving (8x14)=24 in X
RCLx	→	0	2	Indirectly recalls the contents of register 14 (because register 2 contains the value 14) and multiplies that by the contents of X (ie: 3). As register 14 contains 8, the result is 24.
RCLx	.	1	2	Recalls the value of local data register 12 using special dot notation. This can only be used when local data registers have been allocated by the LocR command
RCLx	.	X		Pressing the dot key twice (remember the "X" letter is on the dot key) is an optimisation that selects the X register as the argument. Note this is exactly the same as RCLx followed by ENTER and then the letter "X". When the X register contains 4, this command sequence results in X holding 16.
RCLx	ENTER	Z		Recalls and multiplies the contents of either the Z register or the T (top of stack) register with the X register. The "enter" key switches the calculator into temporary alpha mode allowing the entry of the "T" or "Z" letter.  The "Z" letter is found on the addition key, while "T" is found on the multiplication key.
RCLx	ENTER	T		
RCLx	Y, A, B, C, D, I, L, J and K			If the register name isn't either "X", "Z" or "T" then it can be typed immediately as the first key.

When it comes to STO or RCL, the registers "T", "Z" and "X" present a specific challenge given the letter "T" sits on the multiplication key, "Z" on addition and "X" on the dot key. Without some kind of extra key sequence the calculator would face a potentially ambiguous key sequence. For example, pressing the "X" letter key directly after RCLx would actually indicate an operation on a local register given the letter "X" sits on the dot key). The chart shows that you would instead press the dot key twice to select register "X" as the argument, and use ENTER as a prefix before either "T" or "Z".

### Alternatively you could use ANY register name as follows

RCLx	ENTER	X, Y, Z, T, A, B, C, D, I, L, J and K		An alternative and very easy mechanism to remember is if you want to access any register as an argument to STO or RCL then simply press the ENTER key before pressing <b>ANY</b> register letter
------	-------	---------------------------------------	--	--



## STO and RCL register access - summary

For STO and RCL remember the following:-

1. In order to refer to registers Y, A, B, C, D, I, L, J or K (ie: every register OTHER than "X", "T" and "Z") simply type the register letter immediately after STO or RCL.
2. To refer to registers "T" or "Z" (which are ambiguous because they sit on the math operations multiply and add) prefix the register letter with the ENTER key.
3. To refer to register "X" simply press the dot key twice.

### Note



If you want to simplify this even further then press ENTER before typing ANY register name (which will work for any of the 12 stack registers).

## How tests such as x=? access items in memory

Consider the group of test commands. Tests "x=?" and "x≠?" are secondary functions of the digit 1 key. Additional tests can be found in the **TEST** catalogue and include: x<?, x≤?, x=+0, x=-0, x≈? (ie: approximate equality), x≥? and x>?.

### Note



The "approximate equality" test above is worthy of note. The result will be true when the rounded value of both arguments is equal. In this case the term "round" refers to the calculator rounding using the current display format (FIX, ENG or SCI) or when dealing with fractions rounded using the current denominator.

The tests in this group only permit an immediate argument of either 0 or 1. So for example a test of  $x \neq 2$  isn't allowed. This means that a test against any value other than 0 or 1 must be carried out using a register (*note this is a reasonable limit given the simplicity of embedding a zero or 1 into a command opcode/operand compared to the difficulty of trying to do the same thing for any possible numeric value*). The following chart shows the available options.

Func	1 <sup>st</sup> Key	2 <sup>nd</sup> Key	3 <sup>rd</sup> Key	Comment
X=?	0			Tests the value of X against the immediate value 0, and returns a true or false based on equality
X=?	1			Tests the value of X against the immediate value 1, and returns a true or false based on equality
X=?	Any digit >1. In this case assume 2	3		Tests the value of X against register (in this case) 23 and returns a true or false based on equality
X=?	→	0	2	Indirectly tests the contents of register 14 (because register 2 contains the value 14 (see assumptions list above) and returns a true or false based on equality
X=?	.	1	2	Tests the value of the local data register 12 using special dot notation and returns a true or false based on equality. This can only be used when local data registers have been allocated by the LocR command. <i>For more details on this feature (and dot notation) refer to the local data section in programming.</i>
X=?	.	X		This special optimisation allows you to press the dot key twice to effectively test X against itself. Remember the "X" letter sits on the dot key. This type of function was useful in the past as a test used to determine if X was not a number (NaN) however there are specific NaN tests available in later revisions of the firmware and so this function is now much less relevant
X=?	Y, Z, T, A, B, C, D, I, L, J and K			So long as the register name isn't X, you can type the letter immediately following the test.

As in the case of RCL or STO, as soon as a test is invoked, the calculator opens in temporary alpha mode. Unlike RCL and STO (which has the potential conflict with the addition key ("Z") and the multiplication key ("T"), tests allow you to immediately type any register name including "Z" and "T" but still not "X"...

Alas, for RCL, STO **and** for the test functions, the "X" key has the same ambiguity problem given it sits on the dot key (which is used to indicate a local data address allocated by the LocR catalogue function).

In order to select the "X" register as an argument to a test you again press the dot key twice.

Note that test functions are similar to RCL and STO in the sense that you can also proceed register names with ENTER although this doesn't work with the X register. That said, there won't be many instances where you would want to use X as a test function argument – so it's arguably no big loss.

X=?	ENTER	Y, Z, T, A, B, C, D, I, L, J and K		Alternatively, you can press ENTER to force the WP 34S into temporary alpha mode – which works for any register other than X.
-----	-------	---	--	---

### Tests x=? register access summary

For tests remember the following:-

1. In order to refer to registers Y, Z, T, A, B, C, D, I, L, J or K then simply type the letter immediately after STO or RCL.
2. To refer to register "X" simply press the dot key twice.

#### Note



*If you want to simplify this even further then press ENTER before typing any register name other than X – and while this simplification won't work for register "X" it doesn't matter overly given it is quite unlikely you'll ever need to use X as an argument for a test.*

### How simpler commands such as FIX access items in memory

The "FIX" command used to fix the display to a specific number of digits is worth reviewing because the number of input option combinations are fewer.

Func	1 <sup>st</sup> Key	2 <sup>nd</sup> Key	3 <sup>rd</sup> Key	Comment
FIX	0 to 9			Fixes the display to 0 to 9 decimal places
FIX	→	0	5	Fixes the display to 4 decimal places using indirect addressing to access register 5 (which contains the value 4).

With the FIX command, you cannot for example select a register as an argument. However, there is nothing to stop you employing indirect addressing to set the number of digits required as the second line in the table shows.

### ***Temporary Alpha Layout***

When the calculator enables this mode, the following alpha, numeric keys can be pressed. Note that in this mode the alpha value becomes the primary value (so no shifts are required).

<b>Temporary Alpha Keyboard Keys</b>	<b>Comment</b>
X, Y, Z, T, A, B, C, D, I, L, J and K	Each of these corresponds to the respective RPN stack register and can be found on the grey letter next to each respective key. So for example, T is found on the multiply button, whereas X is found on the decimal point button.  Context will decide when each can and cannot be used as described in the above notes.  The important letters to keep in mind are "T", "Z" and "X" (see above notes).
0 1 2 3 4 5 6 7 8 9	The normal keyboard digits.
→	The right arrow – which is used when employing indirect addressing.
ENTER	Switches the mode on or off
←	Backspace to clear entry characters
EXIT	Cancel the current command

# Part 3

## WP 34S

### Integer Modes

## Integer modes - using alternate bases

If you are familiar with HP calculators of old you may have heard of or even used the HP-16C calculator. This constant memory calculator was aimed at programmers, and came with a rich set of integer and logic operations across different bases. Number representation in the WP 34S is as versatile as the HP 16C with a similar range of functions available to the user.

### Note



Please note that some of the following sections are based on the HP-16C manual but rewritten to cope with this rather different catalogue based calculator.

By default, the calculator starts in floating point mode (referred to as the DECM mode of operation). However, the calculator can easily be switched into integer mode, and in any base from 2 to 16. After you switch the calculator into integer mode, you can revert back to floating point mode anytime by using the key sequence



Switches to common integer bases such as binary (base 2), octal (base 8), decimal (base 10) and hexadecimal (base 16) are available directly on the keyboard



Secondary functions which switches to binary base 2

Secondary functions which switches to octal base 8

Secondary functions which switches to decimal integer base 10

Secondary functions which switches to hexadecimal base 16

### Note



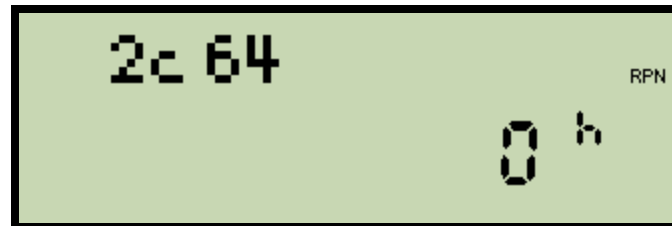
Some of the integer commands used in this section are located in catalogues such as X.FCN - an example being the binary bit shift command SL (shift left) explained in subsequent notes.

These commands only apply to integer modes, and so only appear in the catalogue command list when the calculator is switched into integer mode. In floating point mode, you won't find an SL command.

Switching to other bases (3,4,5,6,7,9,11,12,13,14,15) can be achieved by invoking the BASE command in the MODE catalogue. Let's demonstrate two methods now to switch to and from hexadecimal. First we will use the BASE command.



The display will then show



This display immediately confirms that the calculator is in integer mode by displaying four digits at the top left. "2c" indicates that the calculator is running in 2's complement mode, and the 64 indicates that the calculator is using 64 bits of precision (both are normal calculator defaults).

**Note**



*Hexadecimal numbers use numeric digits 0 to 9 and the alpha characters A, B, C, D, E and F (to represent the decimal values 10 through 15). When the calculator switches to base 16, it automatically interprets the top row of calculator keys as A to F during number entry*

Before we leave integer mode and revert back to floating point, enter the hexadecimal number FF by pressing the F key twice ("F" is the alpha function for the CPX key at the top right of the keyboard). FF in hex is 255 in decimal notation. Hit enter and swap back to floating point mode using the following keystrokes...



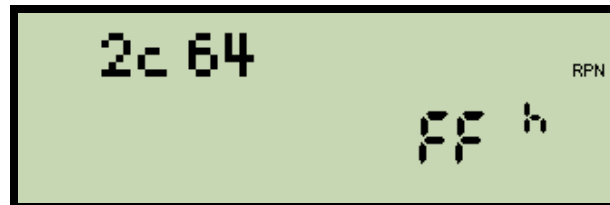
...you will note that the calculator converts (if it can) the contents of the X register to a floating point value and will display...



Switching to one of the four common integer bases (namely binary base 2, octal base 8, decimal base 10 and hexadecimal base 16) is even easier – because shortcuts can be found on the keyboard. Switching to hexadecimal requires the following keystrokes (note that “16” is the secondary function for the EEX key)



And again, note how the calculator will attempt to convert the contents of the X register into a number suited to the new hexadecimal base.



Let us now have a look at some of the functions we will use in order to control the base and also the arithmetic that can be performed in these modes. Open the MODE catalogue and position yourself on the commands starting with B...



If you scroll through the MODE catalogue list using the scroll keys, you should be able to find each of the following commands.

<b>BASE</b>	Switches to a new base for integer calculations where $2 \leq n \leq 16$ . Popular bases (hex, decimal, octal and binary) are available only via the secondary links on the keyboard (see image above).
<b>1COMPL</b>	Sets the calculator into 1's complement mode
<b>2COMPL</b>	Sets the calculator into 2's complement mode
<b>SIGNMT</b>	Sets the calculator into Sign & Mantissa mode (not covered in this guide).
<b>UNSIGN</b>	Sets the calculator into unsigned mode of operation
<b>WSIZE</b>	Defines the number of bits in a full calculator integer mode word. By default this is 64 bit. Note that the command "WSIZE 0" will set the maximum word size of 64 bits

A set of complementary “query” functions can also be found in the TEST catalogue as follows

<b>IBASE?</b>	Queries the current base and leaves the result in the X register
<b>INTM?</b>	Tests to see if the WP34S is working in integer mode. The value is returned in the X register
<b>SMODE?</b>	Queries the current mode of integer operation as follows: 2 when in 2's complement mode 1 when in 1's complement mode 0 when unsigned mode -1 when sign and mantissa mode The value is returned in the X register
<b>WSIZE?</b>	Queries the number of bits in a calculator word leaving the result in the X register

## Complement and unsigned mode

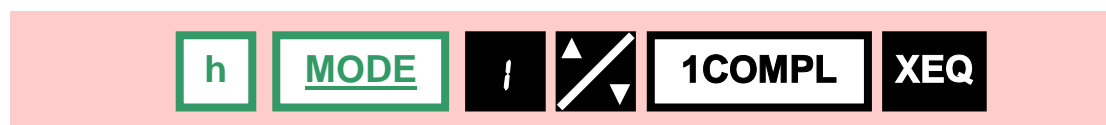
The WP 34S provides four representation modes when dealing with integer math. These are known as 1's complement, 2's complement, unsigned modes and Sign and Mantissa mode<sup>\*1</sup>.

By default, the calculator uses the 2's complement mode but can be switched to any other mode by using the 1COMPL, 2COMPL, UNSIGN or SIGNMT commands in the MODE catalogue. All examples in the following notes will use 2's complement (unless indicated otherwise).

In a binary representation of a number with a sign (-ve or +ve) the most significant bit serves as the sign bit, 0 for positive, and 1 for negative. Decimal mode adds a refinement where a negative number is shown with a minus sign.

### 1's Complement Mode

The following sequence will switch to 1's complement mode of operation

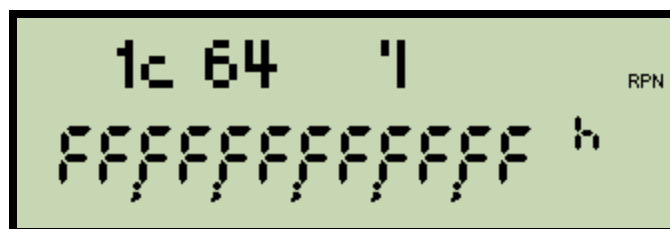


Note how the display now shows "1c" at the top left to confirm the switch to 1's complement mode. Clear the display (using the back space key).

When you click the [+/-] key (change sign) the calculator takes the 1's complement of the contents of the X register which involves complementing or inverting all bits. Hence the 1's complement value of zero is FFFFFFFFFFFFFFFF (remember the calculator is set to use a 64 bit number representation).

The 1's complement number system allows an equal number of both positive and negative numbers, but with the ambiguity of having two representations for zero (0 or -0).

After this key sequence the display will show the following



You may have noticed that this display is actually missing four hex digits (there should be 16 in a 64 bit representation), and that a new notation involving two vertical bars (on the top line, to the centre right of the display) has appeared. To make sense of this display we need to review how the WP 34S displays long numbers.

<sup>\*1</sup> Note that Sign and Mantissa mode are not covered in this guide



## Viewing long number displays – introduction

The display of the long number in the above example highlights a particular problem for any calculator. In this particular example the WP 34S has the problem of trying to display 16 digits using a 12 digit LCD display. To do this, it employs a special display notation involving one or more windows to traverse all digits of the full number.

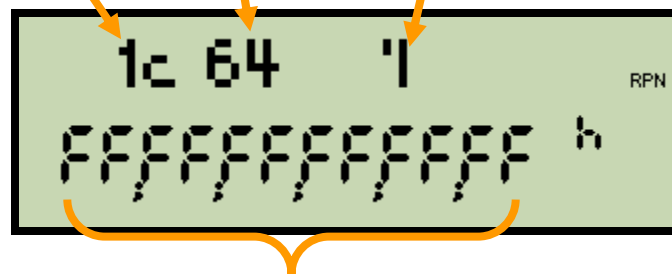
"1c" confirms the calculator is in 1's complement mode...

...working with 64 bits of precision

Two vertical bars indicate that this number requires a total of two display windows to show.

The long bar (right hand) confirms that the WP 34S is displaying the least significant 12 digits of the number (in the 1<sup>st</sup> of 2 windows).

The short bar (left hand) confirms that the most significant digits window is currently hidden.



12 of the 16 digits are displayed here with the least significant nibble shown on the far right

The two vertical bars define the limits of the display problem. They confirm, in this case, that the calculator will display the number using a total of 2 windows (because there are two vertical lines). As the longer of the two vertical lines is currently on the right hand side, this means that the current window (1 of 2) is currently showing the right hand side of the full number (namely the least significant portion). The shorter left hand vertical line means that the left hand of the full number (namely the most significant portion) is hidden from view.

We press the left horizontal scroll key to jump to the window holding more significant bits ◀ or the right horizontal scroll key ▶ to jump to the window holding less significant bits.

Note that both scroll keys are secondary functions under the X exchange Y key ie:

**x↔y**

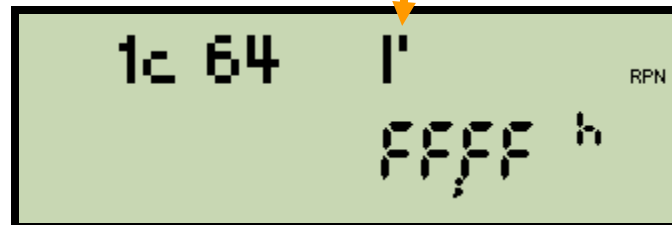
In move to the next window – showing the most significant digits use...



The display will then change as follows:-

Note how the long bar is now on the left which means that the display is now showing the most significant digits.

The short bar on the right means that the least significant digits are now hidden



You can swap back in order to see the least significant digits by...



**Note**



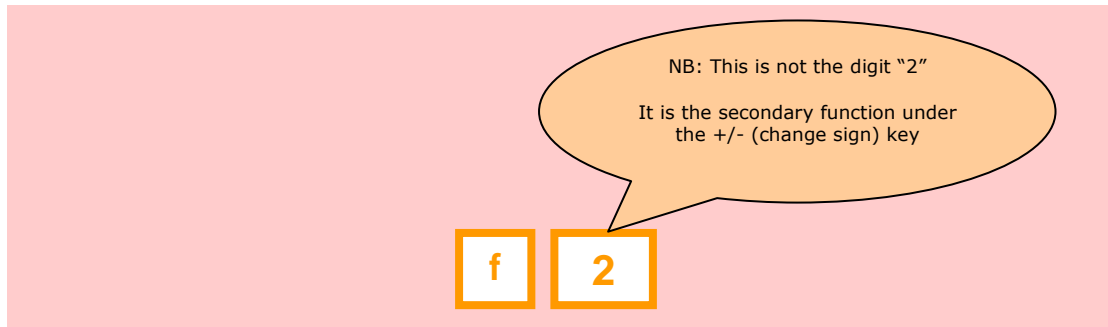
Hexadecimal numbers use numeric digits 0 to 9 and the alpha characters A, B, C, D, E and F (to represent the decimal values 10 through 15). When the calculator switches to base 16, it automatically interprets the top row of calculator keys as A to F during number entry

## Viewing very long 64 bit binary numbers

Let us now extend this example to see how the calculator deals with very long number displays. With the calculator unchanged from above, clear the display (click the backspace key) and then type in the rather long 48 bit hex number

**123456789AbC**

Click enter, and switch to binary base 2 either by using the shortcut key sequence below (or by opening the MODE catalogue, scrolling to the BASE command, executing it and then entering the value 2).



The display will switch to base 2, and will show the following



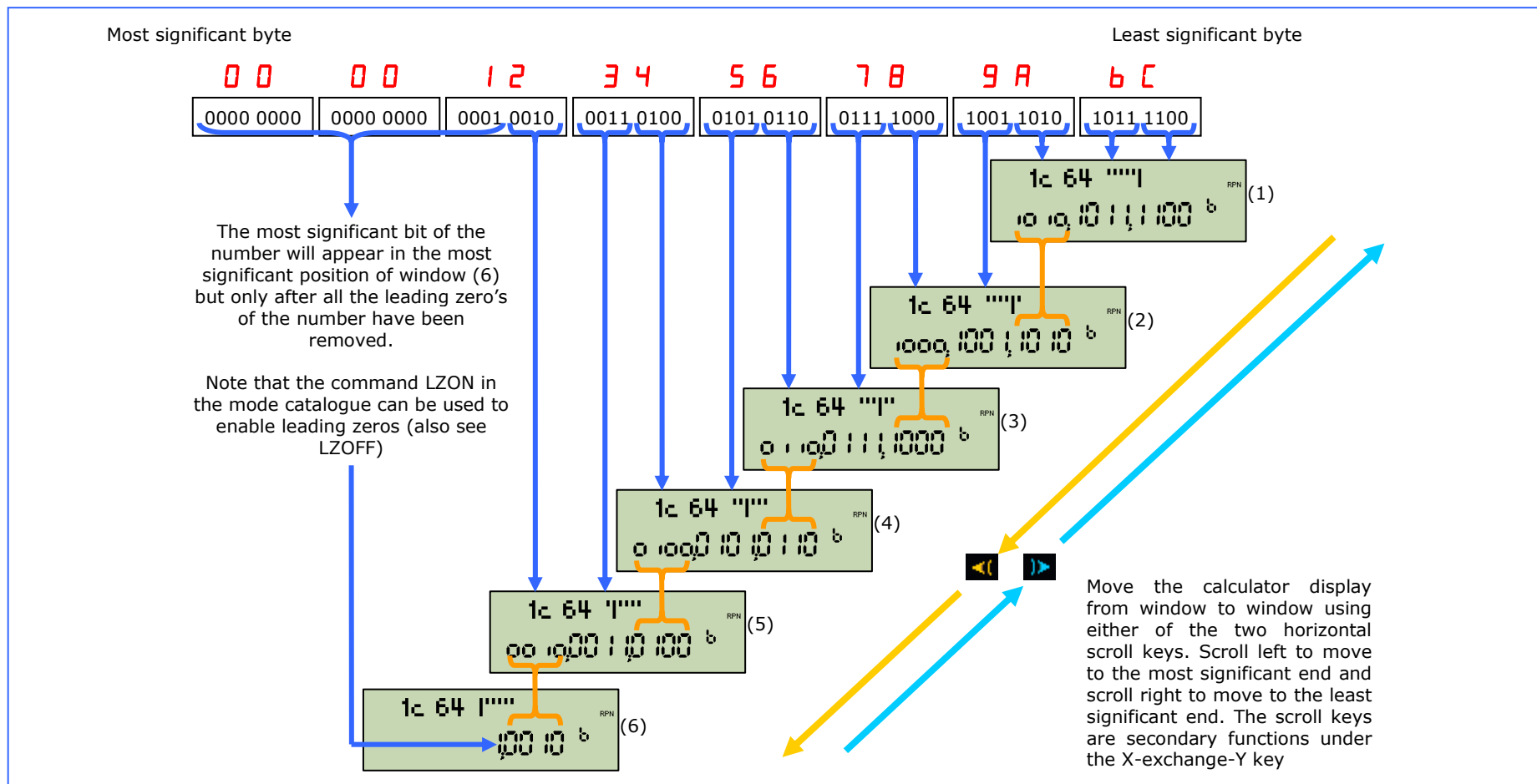
With this number the display has to cope with the problem of displaying 45 bits of binary (once all the leading zeros have been removed) on a 12 character display.

Just as in the above hex example it breaks the full number down into the smallest number of display windows it can – where in this case each window consists of 12 bits.

The 6 vertical lines confirm that there will be 6 windows, and the fact that the right hand vertical line is the longest means that this particular window above is showing the least significant 12 bits of the full number.

You will notice that the display deemphasises the most significant 4 bits (using small characters) because these four bits will appear again in the next window in the sequence of 6. The "overlap" of four bits is designed to assist readability by giving the user a hex digit (4 bits) key in common with any two adjacent windows.

Let us now look at the entire hex number when placed side by side with the six display windows.



Observe that 4 bits of overlap are always used between each display window when showing binary (see the orange links). This is done to assist readability. However it can also be exceptionally confusing. (See note regarding the subtle feature of this display below)

In the above diagram observe that the most significant zeros are all removed from the display (just as per any normal number display) depending on the use of the leading zero switch commands (in the MODE catalogue - see LZOFF, and LZON). The window displays will only ever show significant digits with the most significant in this case being located in the high nibble of the 6<sup>th</sup> byte – and which is shown in display window (6)

### **Subtle Feature of this display**

There is one subtlety with the binary display mechanism adopted by the WP 34S that is worthy of note for those readers completely puzzled by the purpose of the overlapping four bits (as I was).

The display of base 2 binary is actually unique (compared to the display of other bases on the WP 34S) in the sense that it always displays one byte of 8 bits per window. If the user simply ignores the deemphasised overlapping four bits, then each distinct window shows exactly one byte of the full number. In our example above, if we review each of the six windows, and ignore the deemphasised four bits we see the following byte values...

Display Window	Binary in display window	Value in Hex
1	1011 1100	BChex
2	1001 1010	9Ahex
3	0111 1000	78hex
4	0101 0110	56hex
5	0011 0100	34hex
6	0001 0010	12hex

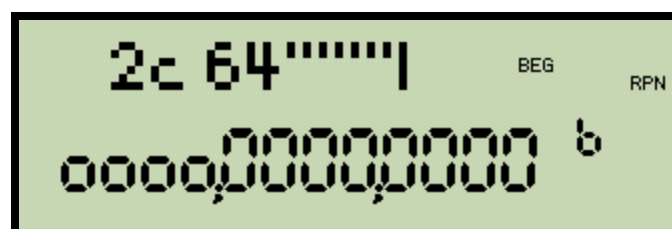
Starting from the most significant byte shown in display window 6 (lowest row) and reading towards the least significant byte in display window 1 gives the full number of 123456789ABC – which was off course the hex number we originally placed in X.

### **Leading Zero's ON / OFF switch**

This switch only applies when the calculator is working in integer mode.

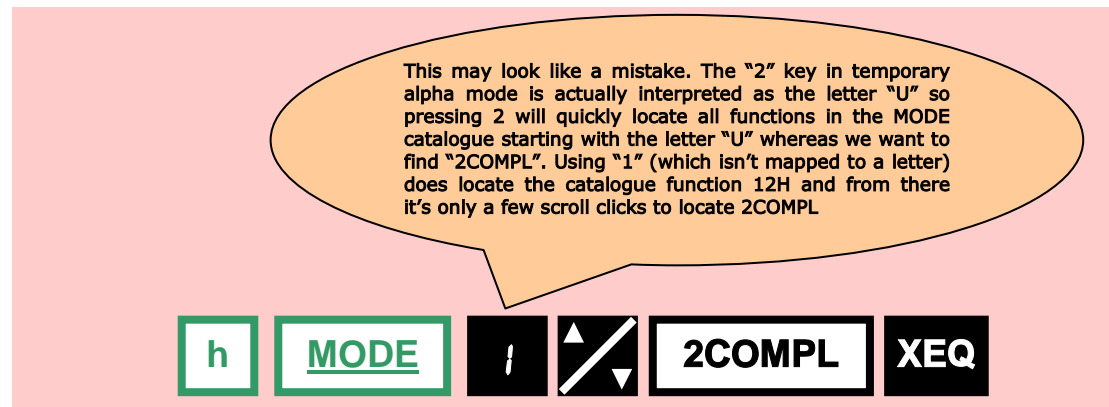
By default leading zeros are suppressed whenever a number is being displayed. Leading zero's can optionally be enabled using the LZON command, or alternatively disabled via the LZOFF command (both located in the MODE catalogue) a feature that can be useful when performing binary logic and/or shift operations on relatively short precision numbers.

The feature becomes more of a hindrance when precision is set to a full 64 bits – and the X register is cleared. In which case you have to cope with the normal long viewing window system in order to see 64 bits of zero – ie:



## 2's Complement Mode

The following sequence will switch to 2's complement mode of operation



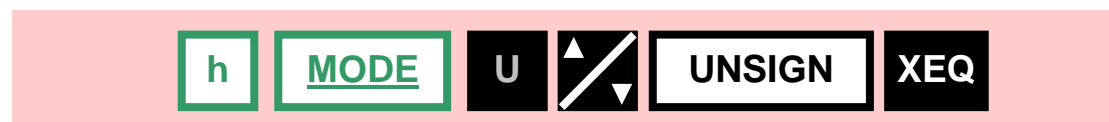
Note how the display now shows "2c" at the top left to confirm the switch to 2's complement node. Clear the display (using the back space key).

When you click the [+/-] key (change sign) the calculator takes the 2's complement of the contents of the X register by complementing or inverting all bits and then adding 1. Hence, the 2's complement of zero is zero.

The 2's complement number system only has one representation for zero, but it always has one more negative number in its representation compared to the number of positive values.

## Unsigned Mode

The following sequence will switch to an unsigned mode of operation when the calculator is working in an integer mode



Note how the display now shows "un" at the top left to confirm the switch to unsigned mode. Clear the display (using the back space key).

Unsigned mode has no sign bit. All bits in the number hold magnitude information so that the display representation of an n bit word is  $2^n$  and the largest value that can be represented is  $2^n - 1$

Changing signs in an unsigned mode doesn't make any sense. If you press the [+/-] key in an unsigned mode the calculator will take the 2's complement of the value in the X register which will effectively overflow the notation system being used. Consequently – the "o" annunciator in the display will show.



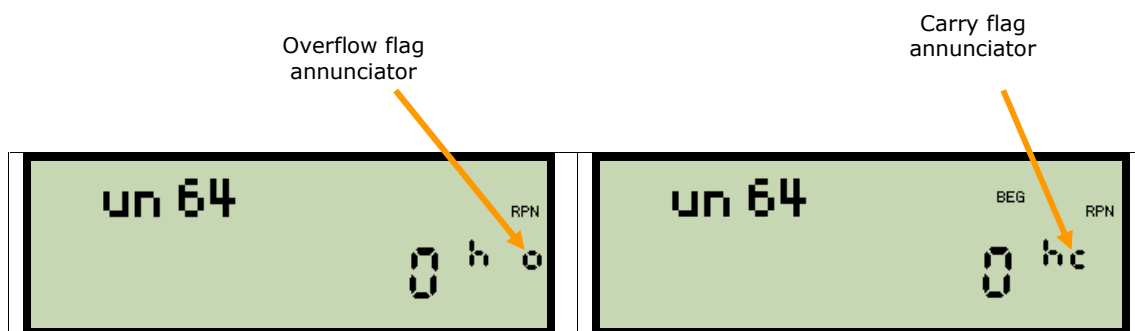
Using a word size of 4, the following table illustrates how the various complement modes affect the decimal representation of all possible values.

Binary	1's Complement	2's Complement	Unsigned
0111	7	7	7
0110	6	6	6
0101	5	5	5
0100	4	4	4
0011	3	3	3
0010	2	2	2
0001	1	1	1
0000	0	0	0
1111	-0	-1	15
1110	-1	-2	14
1101	-2	-3	13
1100	-3	-4	12
1011	-4	-5	11
1010	-5	-6	10
1001	-6	-7	9
1000	-7	-8	8

## Bit wise operations and integer math

Bit wise operations and arithmetic functions can and will result in a carry, and/or overflow condition – which the WP 34S reports using two flags on the right hand side of the display.

An example of each is shown below



The carry and overflow flags are directly addressable in the WP 34S, so you can manipulate them directly using either the set flag (SF) or clear flag (CF) commands (secondary functions of the scroll down key).

The flag naming conventions on the calculator share the set of special names used by the stack registers (ie: X, Y, Z, T, A, B, C, D, I, L, J and K) and so carry has the notation "C" while overflow is designated "B". Relevant flags here are named as follows:-

Flag name	Description
A	Equals flag – when asserted illuminates the "=" annunciator in the display.
B	Overflow flag – when the result of a calculation or operation causes a result that exceeds the number representation.
C	Carry / borrow flag. Carry when performing addition, and borrow when dealing with subtraction. Some shift and rotate operations use this flag in integer mode.
D	Danger flag – may be used to allow special exception results for example infinity and non numeric (NaN) without getting an error (the system reads this flag)

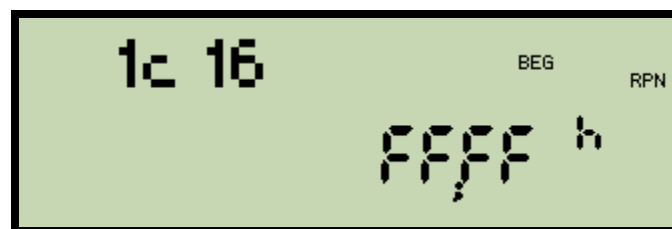
For example – the following manual key sequence will cause the “=” annunciator in the display to appear...



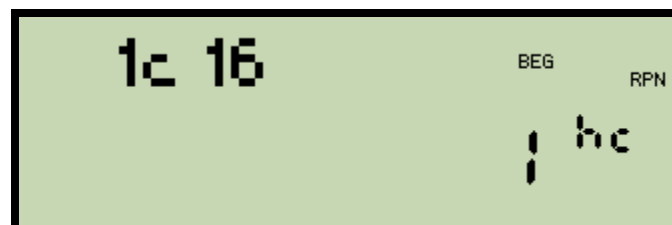
and this sequence will remove the equal's annunciator by clearing the flag...



Consider the following two operations. If we set the display to have 16 bits of precision, using base 16, 1's complement and enter the hex value FFFF we get the following display



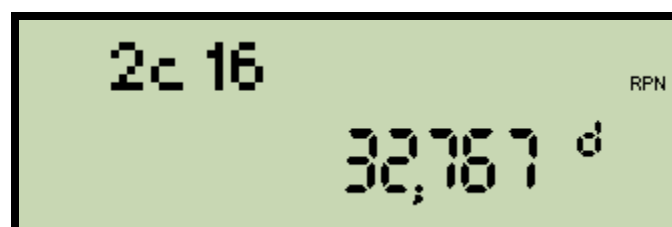
If we then add 1 we would expect to generate a carry bit. As a result we see the following display



Clear the carry flag now using...

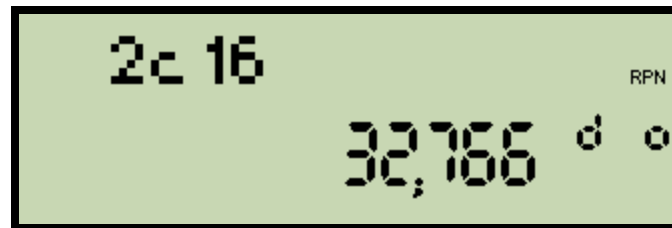


Switch to 2's complement notation with 16 bits of precision. Set the base to 10, and enter the value 32767 which will result in the following display





Now multiply by 2 – which will cause the number system to overflow. The resulting display is as shown below. Note the overflow flag (small “o” on the far right)



Clear the overflow flag now using (g, CF B)



### ***Integer arithmetic functions***

For the following examples – set the calculator into 2’s complement mode, with 16 bits of precision using base 16 (hexadecimal)

First – set the calculator into 2’s complement mode.



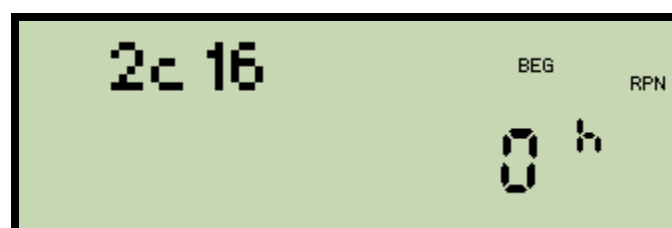
Set 16 bits of precision



Switch into integer mode



You should now see the following display



## Addition, Subtraction, Multiplication and Division

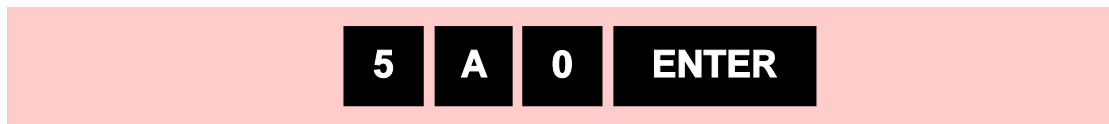
These four arithmetic operations can all be performed using integer math regardless of base. The operation as far as the RPN stack is concerned works exactly the same way as for floating point (as described in detail above).

The following points are worth keeping in mind.

- Division in integer mode will truncate the fractional part of the quotient.
- Any arithmetic operation on integers except multiplication will set/clear both the carry flag and the overflow flag. Multiplication will only affect the overflow flag.
- Multiplication of two  $n$  bit values requires a maximum  $2n$  bit result space.
- You can swap bases anytime you please (something the following examples demonstrate)

**Example:** Find  $(5A0_{16}) / (177764_8)$  using 2's complement and 16 bits of precision. Give the answer in hexadecimal

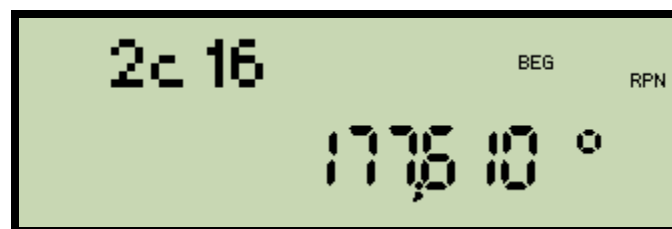
As we are working in base 16, we can enter the first value easily



Now, because the next number  $(177764_8)$  is an octal number – swap the base before entering it into the X register, and then pressing the divide key.



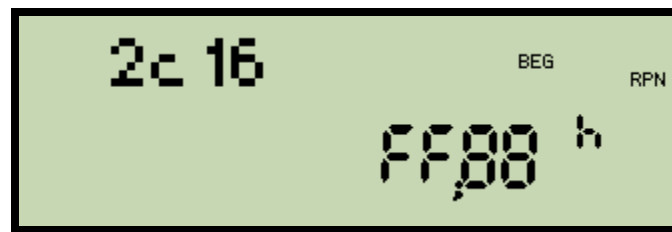
At that point, we have our result – but the current base is octal.



To convert the answer to hex, simply swap back to base 16



At which point the display will become...



The answer is correctly shown as:  $FF88_{16}$

### Addition and Subtraction in 1's complement mode

In 2's complement mode, and unsigned mode, the result of an addition or subtraction is simply the sum or difference between the X and Y register contents. In 1's complement notation however, the result of addition is dependant on the carry flag, and with subtraction on the borrow flag (effectively the same flag).

If a carry out of the most significant bit occurs, then 1 is added to the result. If a "borrow" into the most significant bit occurs, then 1 is subtracted from the result. Both cases set the carry flag.

*Use 1's complement mode, with four bits of precision & base 10 for all examples below*

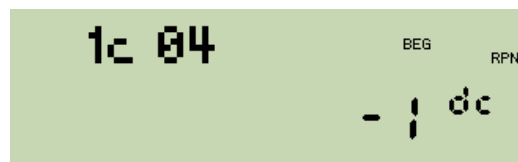
#### Carry Example

$$\begin{array}{r} -1 \\ + (-1) \\ \hline -2_{10} \end{array} \quad \begin{array}{r} 1110 \\ +1110 \\ \hline 1100 \\ + 1 \\ \hline 1101 \end{array}$$



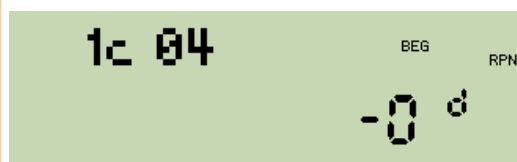
#### Borrow Example

$$\begin{array}{r} 3 \\ -4 \\ \hline -1_{10} \end{array} \quad \begin{array}{r} 0011 \\ -0100 \\ \hline 1111 \\ - 1 \\ \hline 1110 \end{array}$$



#### No Carry Example

$$\begin{array}{r} -3 \\ + 3 \\ \hline -0_{10} \end{array} \quad \begin{array}{r} 1100 \\ +0010 \\ \hline 1111 \end{array}$$



#### No Borrow Example

$$\begin{array}{r} 6 \\ -5 \\ \hline 1_{10} \end{array} \quad \begin{array}{r} 0110 \\ -0101 \\ \hline 0001 \end{array}$$



## The carry flag during addition

Regardless of the complement mode, the carry flag annunciator will be set whenever an operation causes a carry out of the most significant bit of the representation space. If no such carry occurs, the flag will be cleared.

*Use 2's complement mode, with four bits of precision & base 10 for both examples below*

<u>Carry Set</u>	<u>Carry cleared</u>
$\begin{array}{r} -6 \\ + (-4) \\ \hline 6_{10} \end{array}$	$\begin{array}{r} 6 \\ + 1 \\ \hline 7_{10} \end{array}$
$\begin{array}{r} 1010 \\ + 1100 \\ \hline 0110 \end{array}$	$\begin{array}{r} 0110 \\ + 0001 \\ \hline 0111 \end{array}$
<div style="background-color: #d4edda; padding: 10px; border: 1px solid #c3e6cb;"> <div style="display: flex; justify-content: space-between;"> <span>2c 04</span> <span>BEG RPN</span> </div> <div style="text-align: center; margin-top: 10px;"> <span style="font-size: 2em;">6</span> <span style="font-size: 1.5em;">dc</span> </div> </div>	<div style="background-color: #d4edda; padding: 10px; border: 1px solid #c3e6cb;"> <div style="display: flex; justify-content: space-between;"> <span>2c 04</span> <span>BEG RPN</span> </div> <div style="text-align: center; margin-top: 10px;"> <span style="font-size: 2em;">7</span> <span style="font-size: 1.5em;">d</span> </div> </div>
<p>Note that in this example the representation space will be exceeded by the result and so the overflow flag will be set as well as the carry flag</p>	

## The carry flag during subtraction

Regardless of the complement mode, the carry flag annunciator will be asserted whenever a binary subtraction results in a borrow into the most significant bit, otherwise it will be cleared. In the WP 34S subtraction is NOT computed as the addition of a negative number – which affects how carry generation occurs.

*Use 2's complement mode, with four bits of precision & base 10 for both examples below*

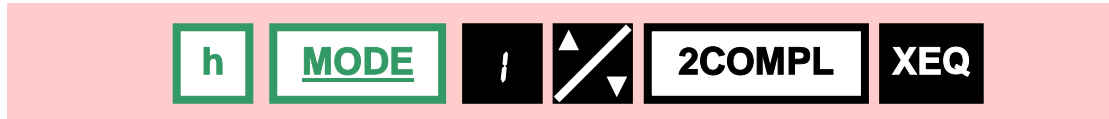
<u>Carry Set</u>	<u>Carry cleared</u>
$\begin{array}{r} -6 \\ - (-4) \\ \hline -2_{10} \end{array}$	$\begin{array}{r} 6 \\ - 1 \\ \hline 5_{10} \end{array}$
$\begin{array}{r} 1010 \\ - 1100 \\ \hline 1110 \end{array}$	$\begin{array}{r} 0110 \\ - 0001 \\ \hline 0101 \end{array}$
<div style="background-color: #d4edda; padding: 10px; border: 1px solid #c3e6cb;"> <div style="display: flex; justify-content: space-between;"> <span>2c 04</span> <span>BEG RPN</span> </div> <div style="text-align: center; margin-top: 10px;"> <span style="font-size: 2em;">-2</span> <span style="font-size: 1.5em;">dc</span> </div> </div>	<div style="background-color: #d4edda; padding: 10px; border: 1px solid #c3e6cb;"> <div style="display: flex; justify-content: space-between;"> <span>2c 04</span> <span>BEG RPN</span> </div> <div style="text-align: center; margin-top: 10px;"> <span style="font-size: 2em;">5</span> <span style="font-size: 1.5em;">d</span> </div> </div>

## Overflow – WP 34S flag B

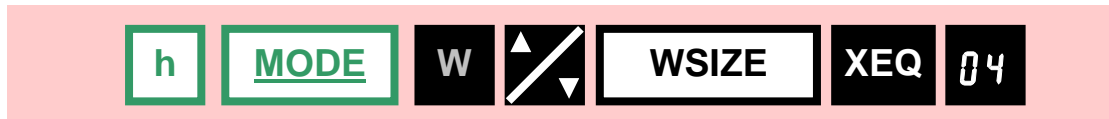
Some arithmetic results that cannot be shown in the current word size and complement mode cause the overflow flag to assert. In the case of division this condition will only occur when in 2's complement mode – specifically when the largest possible negative number is divided by -1

Example: With a word size of 4 bits using 2's complement mode, calculate  $7 + 6$  in base 2, and observe the results of the carry and overflow flags

Set 2's complement mode



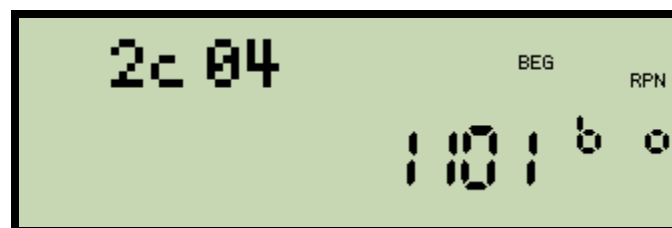
Set 4 bits of precision



Switch into integer mode base 2



Enter the value 111, then type 110, and press + and the display will show the value -3, but where the overflow flag is asserted (note that the carry is cleared).



## Remainder after division and RMDR

Division leaves the integer portion of the result in the X register. If the remainder is not zero, then the carry flag will be set otherwise it will be cleared.

There are times when it can be useful to obtain the remainder, instead of the quotient of the division – which can be done using the RMDR function (the secondary function of the / (divide) key). This function computes  $|Y| \text{ MOD } |X|$ . The sign of the result matches the sign of the dividend (ie: the sign of register Y).

Set the calculator into 2's complement mode, with 16 bits of precision working in base 16

Key strokes	Display	Description
66 ENTER	66h	
7 /	7h	Note the carry annunciator is asserted because 66/7 leaves a non zero remainder
2 /	2h	Note carry is NOT asserted because E/2 leaves a zero remainder
4 <b>h RMDR</b>	3h	Remainder of 7 / 4

## Square root

The square root function computes the root of the value in the X register. The fractional part of the square root is lost – but if the fraction is not zero then the carry is set, otherwise it is cleared.

## Negative Numbers – changing signs

The +/- function will change the sign forming the 1's or 2's complement of the value in the X register. If the X register holds the largest possible negative number in 2's complement mode then the only effect of using this function will be to set the overflow flag.

In unsigned mode, using the +/- key forms a 2's complement of the X register, and also sets the overflow flag (confirming that the notation cannot cope with a negative number).

To enter a negative number, type the number and then click the +/- key.

## Negative Numbers – absolute value

The absolute key on the WP 34S keyboard is the  $|x|$  secondary function of the zero key. This function converts the X register into its absolute value, forming the 1's or 2's complement of a negative number. There is no change if the calculator is in unsigned mode, or the number is positive.

If the X register holds the largest possible negative number in 2's complement mode, then the only effect of taking the absolute value will be to set the overflow flag.

## Logical operations

The suite of logical operations include NOT, OR, AND, XOR available on keys on the keyboard. Addition functions for NOR, NAND and XNOR are available in the X.FCN catalogue. All these functions return the result of a bit to bit operation

OR, NOR, AND, NAND, XOR and XNOR all operate on matching bits in both the X and Y register, leaving their result in X (and the stack drops by 1). The NOT (invert) operator acts on the X register only (no stack drop)

Set the calculator to 2's complement mode, 8 bits of precision & base 2

### Logical NOT

The logical NOT operator simply inverts each bit in the representation. It is the same as taking the 1's complement. Observe that NOT is a one value function.

Example: Enter 10111110 into the X register and use the NOT operator to invert each bit, and obtain 1000001.

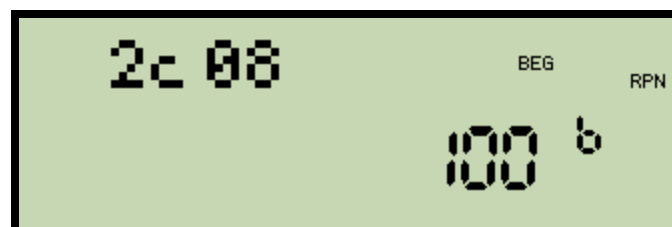


### Logical AND

Logical AND (sometimes called the product) compares two bits, and returns a 1 when both input bits are 1. Otherwise it returns a zero. The following will demonstrate the idea. Note that AND, NAND, OR, NOR, XOR and XNOR are all two value functions.

Y Reg:	0100 1100	<b><u>LOGICAL BITWISE AND</u></b>
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
X Reg:	1011 0110	
<hr/>		
X AND Y:	0000 0100	
<hr/>		

With the calculator in the same mode, enter 1001100, then type 10110110 and select the logical AND function (a secondary function of the 7 key). Giving...



## Logical NAND

Logical NAND (inverse of AND) compares two bits, and returns a 0 when both input bits are 1. Otherwise it returns a 1. The following will demonstrate the idea. NAND is a two value function located in the X.FCN catalogue.

Y Reg:	0100 1100	<b><u>LOGICAL BITWISE NAND</u></b>
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
X Reg:	1011 0110	
<hr/>		
X NAND Y:	1111 1011	
<hr/>		

With the calculator in the same mode, enter 1001100, then type 10110110 and select the logical NAND function in the X.FCN catalogue. Giving...



## Logical OR

Logical OR (sometimes called the sum) compares two bits, and returns a 1 when either or both input bits are 1. If both bits are zero, it returns a zero.

Y Reg:	0100 1100	<b><u>LOGICAL BITWISE OR</u></b>
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
X Reg:	1011 0110	
<hr/>		
X OR Y:	1111 1110	
<hr/>		

Enter 1001100, then type 10110110 and select the logical OR function (a secondary function of the 8 key). Giving...





## Logical NOR

Logical NOR (inverse of OR) compares two bits, and returns a 0 when either or both input bits are 1. If both bits are zero, it returns a 1. NOR is a two value function in the X.FCN catalogue.

Y Reg:	0100 1100	<b><u>LOGICAL BITWISE NOR</u></b>
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
X Reg:	1011 0110	
<hr/>		
X NOR Y:	0000 0001	
<hr/>		

Enter 1001100, then type 10110110 and select the logical NOR function in the X.FCN catalogue. Giving...



## Logical XOR

Logical XOR (sometimes called the difference) compares two bits, and returns a 1 when the two bits are different. If both input bits are the same (either both 0 or both 1) the function returns a zero

Y Reg:	0100 1100	<b><u>LOGICAL BITWISE XOR</u></b>
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
X Reg:	1011 0110	
<hr/>		
X XOR Y:	1111 1010	
<hr/>		

Enter 1001100, then type 10110110 and select the logical XOR function (a secondary function of the 9 key). Giving...

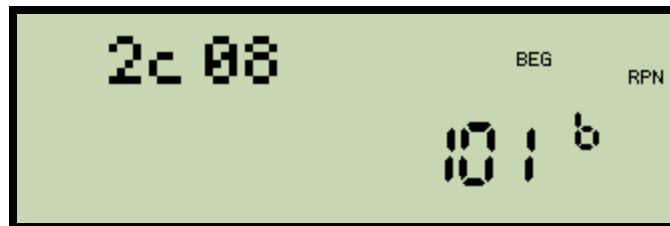


## Logical XNOR

Logical XNOR (inverse of XOR) compares two bits, and returns a 0 when the two bits are different. If both input bits are the same (either both 0 or both 1) the function returns a 1. XNOR is a two value function in the X.FCN catalogue

Y Reg:	0100 1100	<b><u>LOGICAL BITWISE XNOR</u></b>
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
X Reg:	1011 0110	
<hr/>		
X XNOR Y:	0000 0101	
<hr/>		

Enter 1001100, then type 10110110 and select the logical XNOR function in the X.FCN catalogue. Giving...



## Bit Shifting and Rotating

Bit shifting is used to move bits either left or right across positions in a register. The bit that falls out at the end of the word, and also the value of the bit entering the vacated position depends on what type of operation is being performed.

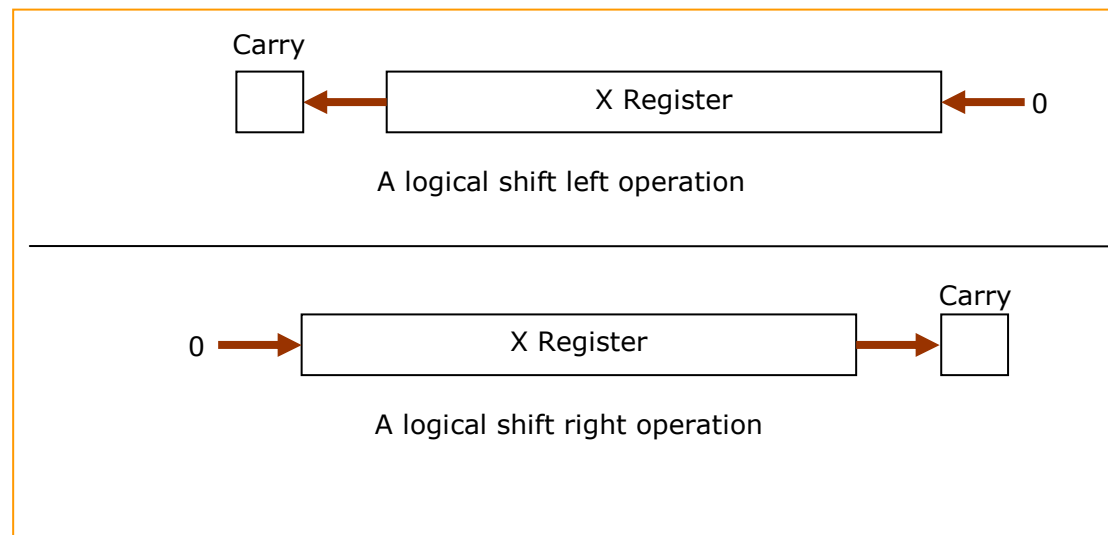
Bit shifting is a very important concept in hardware engineering. Shifting a binary register left  $n$  places has the same effect as multiplying the original register value by  $2^n$ . Similarly, shifting to the right divides. Bit shifting offers significant speed advantages over multiplication/division algorithms even if they only work to powers of 2.

## Shifting Bits

The WP 34S can perform two different shifts on the X register, known as a logical shift, or an arithmetic shift.

## Logical Shifts

Logical shifts (either left or right) move all the bits in the X register  $n$  bits to the left or the right. Bits shifted out of the word overwrite the carry bit and new bits fed into the register are always zero's ie:



The logical shift left command is located in the X.FCN catalogue, and is called SL. Similarly the logical shift right command is in the same catalogue and is called SR. Both commands require an argument of  $n$  (for the number of bit shifts). A zero value makes no change (the calculator executes a no operation - NOP).

### Note



*Remember that these commands only apply to integer modes, and so will only appear in the X.FCN catalogue listing when the calculator is switched to an integer mode of operation.*

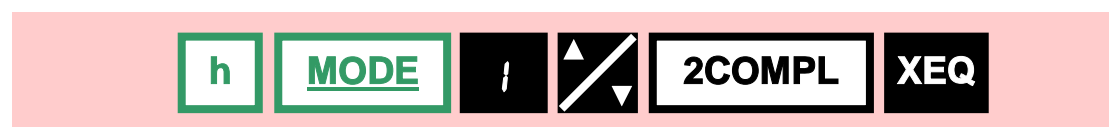
## Left and Right bitwise Justification

A bit pattern can be either left or right justified within the confines of the current word size using commands in the X.FCN catalogue called LJ for left justifications, and RJ for right justifications.

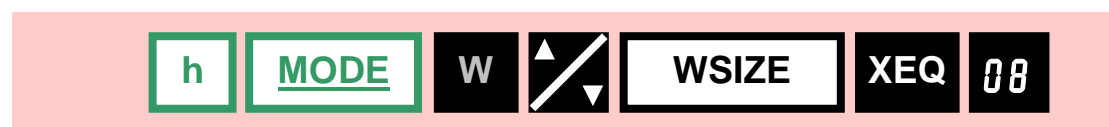
When these commands execute, the stack will lift placing the justified word in the Y register, and a count of how many bits the word had to be moved in order to complete the justification in the X register. The carry flag is not affected by this operation.

Set the calculator into 2's complement notation, with 8 bits of precision using binary base 2.

First – set the calculator into 2's complement mode.



Set 16 bits of precision



Switch into integer mode using base 2



Now type the binary value 1111 into the X register, and select the command LJ in the X.FCN catalogue. After execution, the X register will hold the value 100<sub>2</sub> which indicates that the original bit pattern required shifting 4 places to the left in order to carry out the left justification. Pressing the roll down key to move Y into X reveals the justified value of F0<sub>16</sub> as the following sequence shows

2c 08      BEG      RPN

100 b

← After execution of LJ the X register shows the number of places required to shift left to left justify - in this case 4 were necessary

**R↓**

2c 08      BEG      RPN

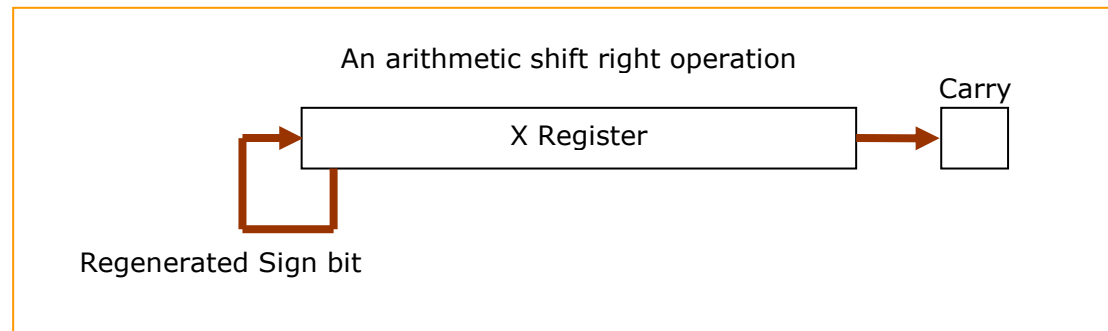
111 0000 b

← The Y register holds the justified value and so can be read by rolling the stack down revealing the left justified value of F0<sub>16</sub>.

The same logic applies to the right justification command RJ

## Arithmetic Shift Right

The function ASR (located in the X.FCN catalogue when the calculator is in integer mode) will move the contents of the word in the X register  $n$  bits to the right (just like SR). However, instead of feeding a zero into the space on the left side of the register the sign bit is regenerated (copied). This doesn't make any difference in unsigned mode which has no sign bit and where ASR behaves like SR. In all cases, the least significant bit in the X register is shifted into the carry bit.



Shifting a positive binary number right  $n$  places is the same as dividing by  $2^n$ . Given the ASR function regenerates the sign bit it can be used to divide an even negative number by 2 (odd numbers actually result in one less than division by 2)

Assume the calculator is using 2's complement notation, with 8 bits of precision, using binary base 2 then entering 0111 1111 ( $127_{10}$ ) into the X register and executing "ASR 1" will result in the carry flag being set, and the X register containing 0011 1111.

Leaving the mode as 2's complement, with 8 bits of precision then entering 1000 0010 ( $-126_{10}$ ) into the X register and executing "ASR 2" will result in the carry flag being set, and the X register containing 1110 0000 ( $-32_{10}$ ).

If we switch to unsigned operation (using UNSIGN in the MODE catalogue) with 8 bits of precision, then entering the same value 1000 0010 ( $130_{10}$ ) into the X register and executing "ASR 2" will result in the carry flag being set, and the X register contains 0010 0000 ( $32_{10}$ ) – in other words, when working in unsigned mode, ASR does indeed behave like a logical shift right (SR) command.

### Note



*Although Sign and Mantissa integer mode is not specifically covered by this guide, it might be useful to know that the ASR command behaves slightly differently in that mode. ASR leaves the top bit unchanged and shifts the rest of the bits right placing a zero into the second top most bit. Consider the value 1000 0010 (which is  $-2_{10}$  in sign and mantissa mode). Executing ASR 1 will result in the X register holding the value 1000 0001 ( $-1_{10}$ ). Observe that the sign bit doesn't move.*

## Rotating bits

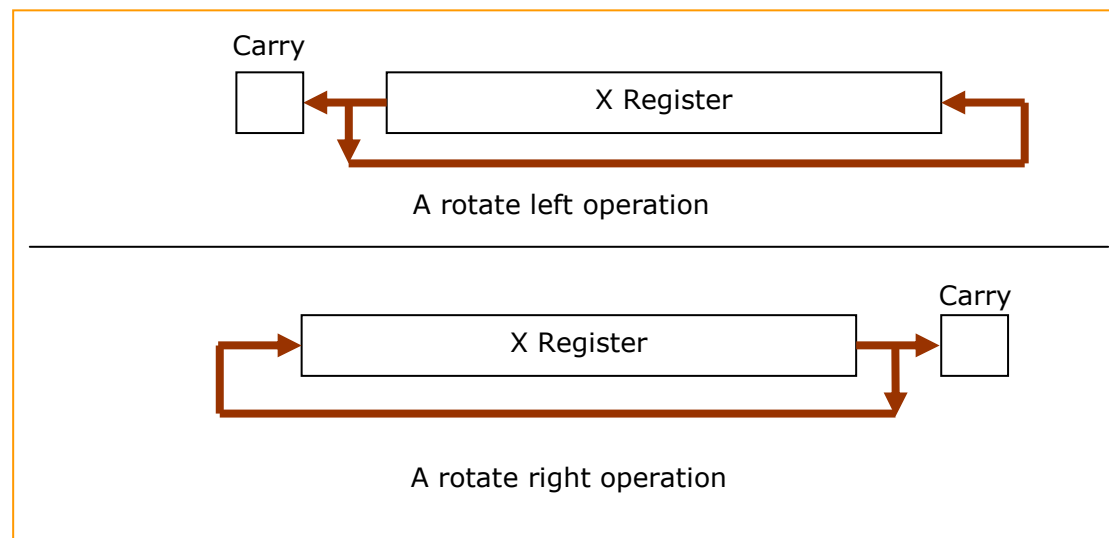
On the WP 34S there are two different types of rotate functions

- Rotate  $n$  places left or right (RL  $n$ , RR  $n$  – both in the X.FCN catalogue)
- Rotate  $n$  places through the carry left or right (RLC  $n$ , RRC  $n$  – both functions in the X.FCN catalogue)

## Rotation

Using the RL n and RR n commands located in the X.FCN catalogue rotate the X register either left or right by n places. These functions differ from the shift commands because the output side of the register is fed to the carry AND to the input side of the same register.

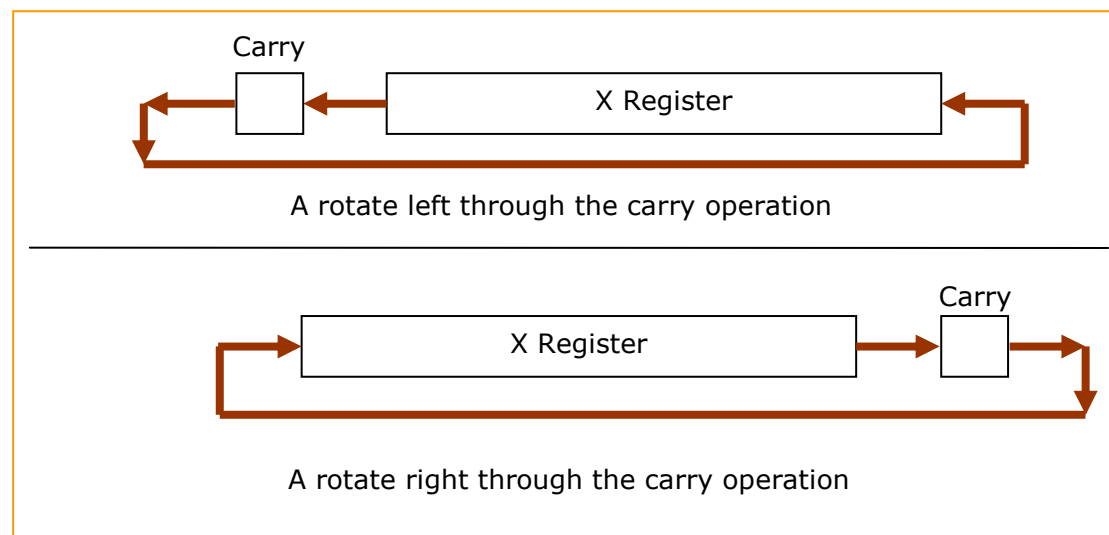
Pictorially RL and RR work as follows



## Rotation through the carry flag

The RLC n and RRC n commands only differ from RL and RR in the way the carry is used as a staging bit.

Pictorially RLC and RRC work as follows

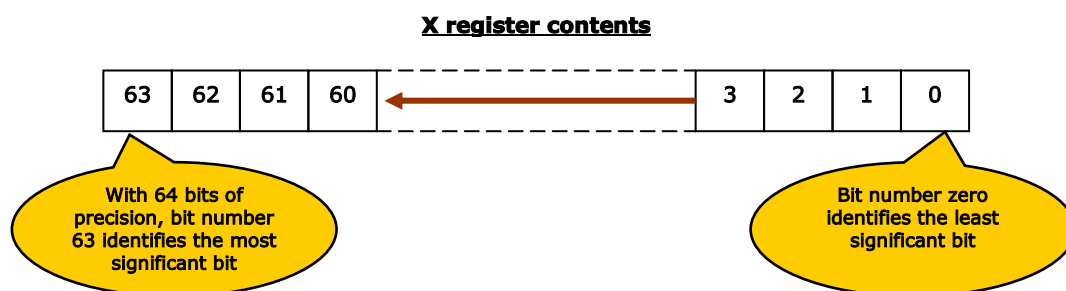


## Negating, Asserting and testing the state of bits

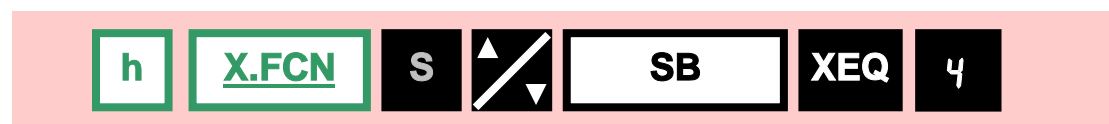
Individual bits in the word in the X register can be asserted (set to 1), or negated (cleared to 0) using the set bit (SB) and the clear bit (CB) commands located in the X.FCN catalogue.

Both commands take an argument to defines the particular bit number to change (ie: set or clear) – a number that starts from zero for the least significant bit of the X register. With n bits of precision in the current calculator word, the most significant bit would be identified by n-1.

In the example below, the calculator is set to 64 bits of precision

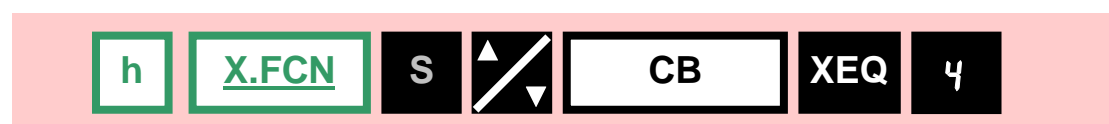


Clear the X register and use the following command sequence to set bit 4.



After executing this command, the X register zero will be changed to  $10000_2$

Execute this sequence to clear the same bit



After executing this command, the X register will be zero.

## Testing bits

Individual bits can be tested using two commands resident in the TEST catalogue. The: **BC? n** command tests to see if bit n is clear in register X. If it is, the function returns a true, otherwise it returns a false

Similarly the command: **BS? n** in the same catalogue tests to see if bit n is set in register X. If it is, the function returns a true, otherwise it returns a false

### Note

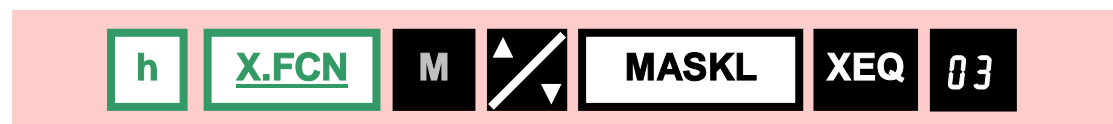


*Remember these commands will only be present in a catalogue when the calculator is switched to integer mode.*

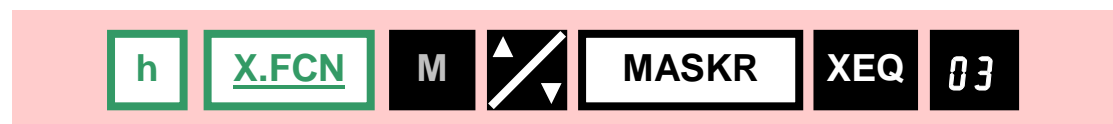
### ***Bit masks – left and right justified***

The MASKL n and MASKR n functions in the X.FCN catalogue are designed to create a left or right justified mask of all bits set to 1 where n defines the number of bits in the mask. These commands can be used to dynamically define a mask that can be used to isolate specific bits in a register by for example AND'ing the mask with the register.

If the calculator is set to 2's complement mode, 8 bits of precision and binary base 2, then executing the following command will create a mask in the X register of  $11100000_2$



Similarly the following command sequence will create a mask in the X register of  $111_2$



### ***Mirroring bits***

The "MIRROR" function located in the X.FCN catalogue when the calculator is working in integer mode takes the bits in the X register working within the word size of the calculator and mirrors them around each other horizontally.

So for example with 2's complement mode, 8 bits of precision and working in binary base 2, a word in the X register of  $00001101$  would become  $10110000$ .

### ***Double Functions***

The WP 34S provides a number of double functions, known as DBLx (double multiply), DBL/ (double divide) and DBLR (double remainder). All three are located in the X.FCN catalogue. These functions allow the precise calculation of a product double the given word size, and the precise calculation of a quotient and remainder from a dividend of double word size.

To obtain meaningful double numbers as results when working in hexadecimal and octal modes, the word boundary (which is based on the number of bits) must not split a digit. You should therefore specify a compatible word size (a multiple of four in Hex mode and a multiple of three in octal mode).



## Double Multiply - DBLx

The DBLx function multiplies two single word quantities – the first in the X register and the 2<sup>nd</sup> in the Y register, leaving the result in both the X and the Y register. With multiplication, the maximum bit depth required for the result is always the sum of the bit depths of the two multiplicands. Note that the stack does not drop – in effect the two original values are overwritten by the product result.

The result is right justified with the least significant bits in the Y register, and the most significant bits in the X register.

With the calculator in 2's complement mode, with 8 bits of precision and using binary base 2 – then consider the following computation involving the multiplication of 100 by 32.

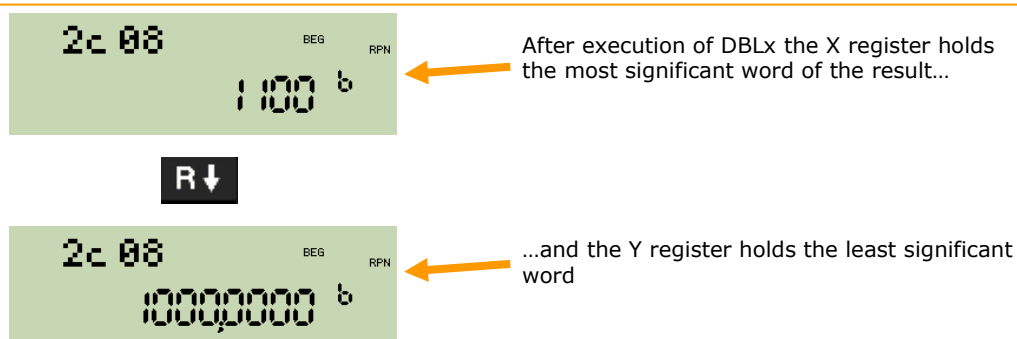
*Use 2's complement mode, with eight bits of precision & base 2 for the example below*

$$\begin{array}{r} 100 \\ \times 32 \\ \hline 3200_{10} \end{array} \qquad \begin{array}{r} 1100100 \\ \times 100000 \\ \hline 1100,1000,0000 \end{array}$$

Switch the calculator into the appropriate modes (2's complement, 8 bits of precision, and binary base 2), then type 1100100<sub>2</sub> and press ENTER. Then type 100000<sub>2</sub> and execute the DBLx command via...



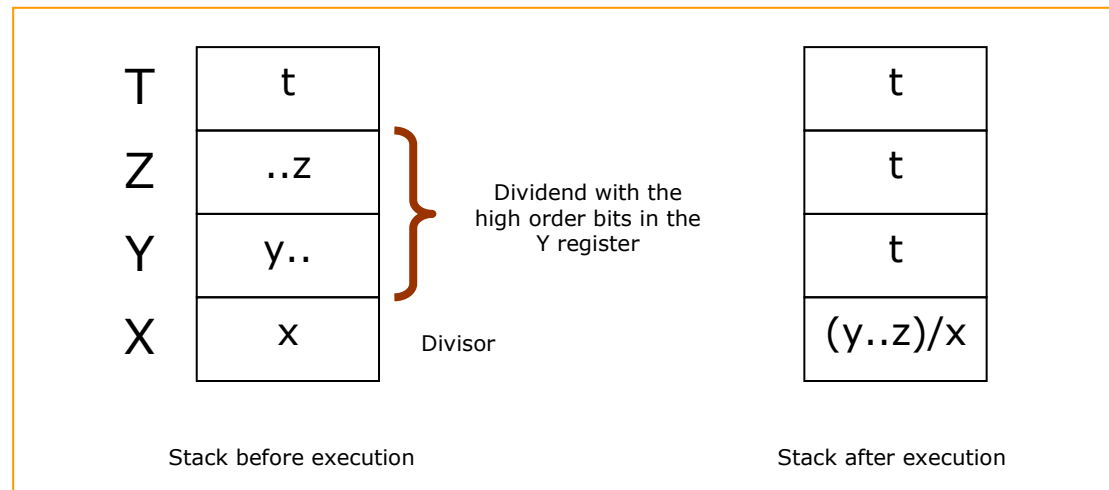
The most significant word of the result will be left in the X register (ie: 1100<sub>2</sub>) and the least significant word of the result will be in Y (10000000<sub>2</sub>) which can be obtained simply by pressing the stack roll down key R↓



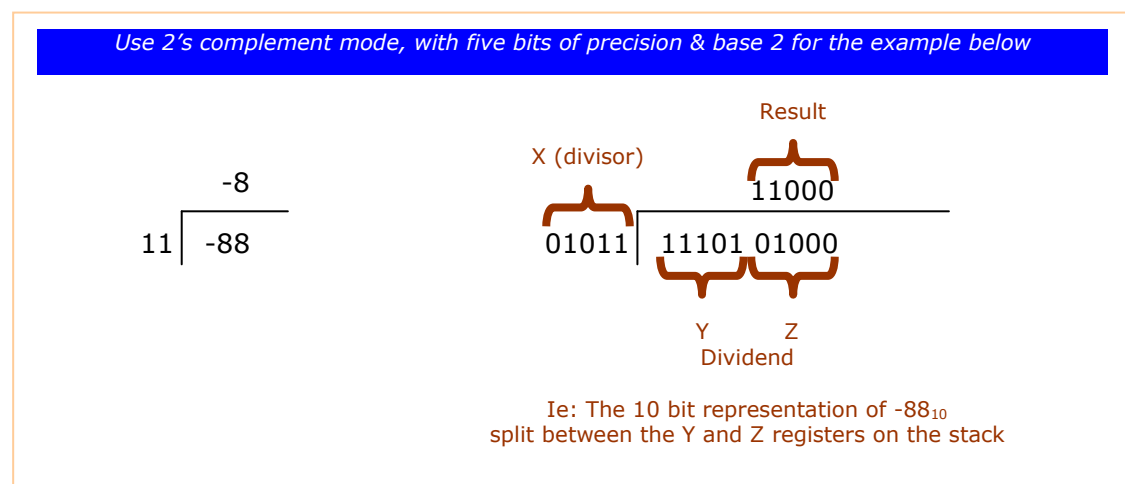
## Double Divide – DBL/

The DBL/ function computes the quotient of a dividend of double word size in the Y and Z registers (where Y holds the most significant bits) divided by a single word divisor in the X register. The stack drops twice during the execution, placing the single word result in the X register.

The stack contents before and after execution are as follows:-



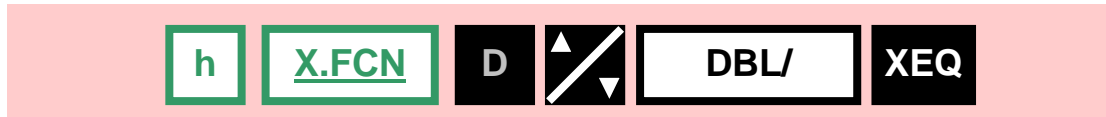
With the calculator in 2's complement mode, with 5 bits of precision and using binary base 2 – consider the following computation ( $-88 \div 11$ ).



Switch the calculator into the appropriate modes (2's complement, 5 bits of precision, and binary base 2), then enter the following keystrokes

Key stroke	Display	Comment
1000 <b>ENTER</b>	0 1000b	These are the least significant bits of the 10 bit dividend – and will end up in the Z register
11101 <b>ENTER</b>	1 110 1b	These are the most significant bits of the 10 bit dividend – and will end up in the Y register
1011 <b>DBL/</b> *	1 1000b	Quotient result (ie: -8)

\* Execute the DBL/ command via the key sequence...



## Double Remainder

The DBLR command operates like DBL/ except that the remainder is returned, instead of the quotient.

The remainder is determined in exactly the same way as for the RMDR command (see earlier notes) with the sign of the result matching the sign of the dividend.

# Part 4

## WP 34S Programming

## Programming

We've now looked at a number of functions – all of which have been operated directly from the keyboard using what we might call an "immediate mode of operation". You type on the keyboard, you get an immediate result.

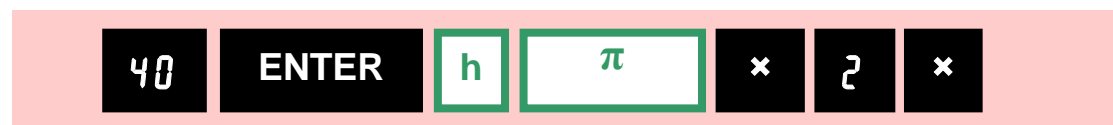
In this section we're going to look at how to write programs for the WP 34S. We are going to cover the following issues.

- How to key in a program
- How to edit a program
- How to run a program
- How to delete a program

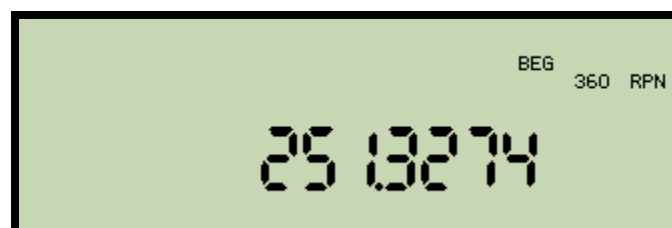
The WP 34S uses a style of writing programs known as Keystroke Programming which mirrors the same key strokes we use when we are using the machine in an immediate mode of operation. Consider the formula used to calculate the circumference of a circle

$$C = 2\pi r$$

Given a radius  $r$  of 40mm, feet, miles or whatever unit you want, you might compute the circumference using the following keystrokes



In this case the radius (40) is entered onto the stack. Pi is then entered and the two are multiplied. Finally 2 is entered and the product of 2 and the intermediate result ( $\pi \cdot r$ ) gives the result. When fixed to 4 decimal places the circumference will be in the X register



We could use this as the basis of a program while keeping two concepts in mind. Firstly the program will need a name (so that the calculator can refer to and call this program uniquely from any others). Secondly it is useful to define the program in such a way that it picks up the radius directly from whatever we enter into the X register before we actually call the program. That way we can use the program to find the circumference of a circle with any radius.

The program would look like this.

01	LBL 'CIR'
02	$\pi$
03	x
04	2
05	x
06	END

You can probably see that when this program runs, it *assumes* that the X register will contain a valid radius (given it starts by loading Pi and then multiplying).

The label (LBL) instruction on line 01 identifies the program uniquely by the name 'CIR'. In this example we are using what is known as a global program label (ie: the string of characters CIR surrounded by single quotes). Global names can be 1, 2 or 3 alpha characters long, and are case sensitive. They are so named because when we later instruct the calculator to execute a "global" program name, the calculator searches for that name throughout the entire available program memory space.

The END instruction on line 06 is necessary to separate this program from the next program in memory. Line 2 enters the value of Pi into the X register, and the multiply instruction on line 3 forms the product of Pi and the radius. Line 4 enters 2, and the multiply instruction on line 5 forms the final result. The program then terminates.

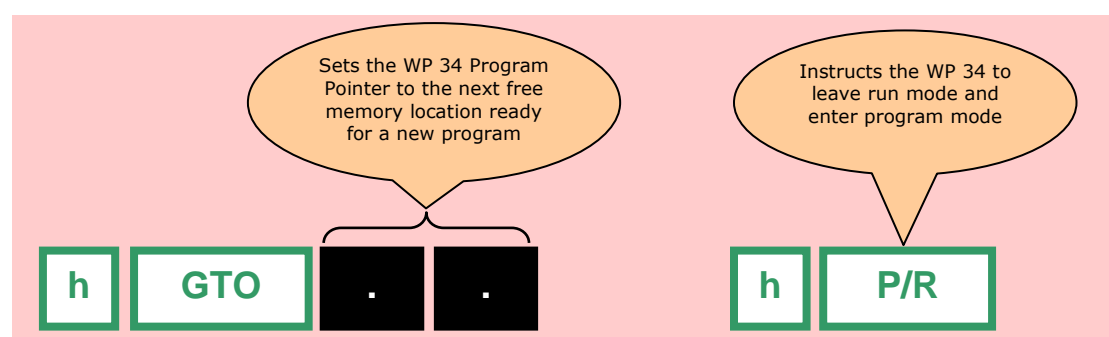
To calculate the circumference, you would key in the radius into the X register and then run the program. At the end of execution, the result (the circumference of the circle) would be found in the X register.

### Keying in our first program – 'CIR'

To key a program into the WP 34S, we start by doing two things.

1. We tell the calculator to set its program pointer to the next free program memory location. We will then store our programs first instruction in that location. The program pointer is used by the calculator to record the address of one particular line of program memory.
2. We then leave "run mode" which is the normal mode for the calculator and enter "Program mode". At that point we can enter the list of instructions.

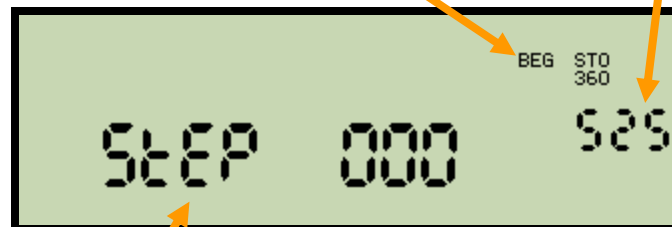
The key sequences required for these two steps are



After the two steps above, you will see something similar to the following display (NB: the free space shown as 525 may differ on your machine depending on the precise allocation of registers etc).

The "BEG" annunciator means that the program is sitting right at the beginning of memory

The number shown here reflects how many program steps are available – but note that this is dependant on how the memory in your calculator has been allocated to registers and so may be different on your machine.



This portion of the display shows what line or step of the program is being displayed.

Let us now enter each program line. If you have already set the calculator into program mode, then simply ignore the first line (below) and work through the remaining key stroke sequences shown on the left side. The right hand shows the screen as it might appear (allowing for the difference in register space on your calculator)

<b>h</b> <b>GTO</b> <b>.</b> <b>.</b> <b>h</b> <b>P/R</b>	BEG ST0 360 STEP 000 529
Now create a new label, and because we are about to enter three alpha characters to define the new global labels name – we use the ENTER key to switch the calculator into the alpha mode. Note that the LBL display will then show a single quote	
<b>f</b> <b>LBL</b> <b>ENTER</b>	LBL' BEG 360 STEP 000 529
Now type in the three letters of the global name. C is on the top line, I is on the roll down key, and R is on the digit 5 key. In alpha mode you DON'T use shifts to input the letters you just press each relevant key.	
<b>C</b>	LBL'C BEG 360 STEP 000 529
<b>I</b>	LBL'CI BEG 360 STEP 000 529
Global (alpha) labels can only be 1, 2 or 3 characters long. As soon as we enter the third and last character "R", the calculator will immediately conclude the label – and add the last single quote. If we wanted to enter a shorter label (ie: 1 or 2 characters) then we would terminate the label name early by pressing ENTER following the last character.	
<b>R</b>	LBL'CIR' BEG ST0 360 STEP 001 523
<b>h</b> <b><math>\pi</math></b>	# $\pi$ ST0 360 STEP 002 522
<b><math>\times</math></b>	$\times$ ST0 360 STEP 003 521
<b>2</b>	2 ST0 360 STEP 004 520
<b><math>\times</math></b>	$\times$ ST0 360 STEP 005 519

Before you press the exit key, note that you can use the two scroll keys ( $\blacktriangledown$   $\blacktriangle$ ) to move up or down the lines of the program. Press exit to leave program mode.

<b>EXIT</b>	360 RPN 00000
-------------	------------------

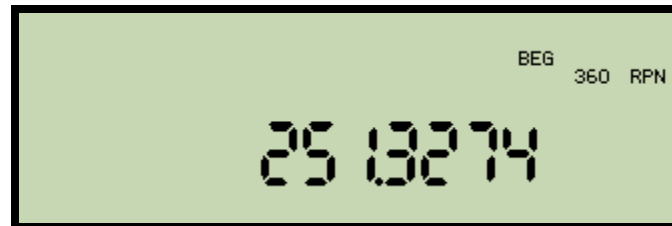


As soon as you press the exit key the calculator automatically inserts an END instruction to correctly terminate this program.

You can now invoke this new program to calculate the circumference of a circle of any radius  $r$ . Key in the radius, and use the XEQ key as shown below.



The program will execute, and the display will show...



Key in 1250, and execute the program again to find the circumference of a different circle with radius 1250 miles.

Now that the program has executed at least one time, you can use a shortcut that takes advantage of the fact that the calculator's program pointer is currently pointing at the CIR program (actually it will be pointing to the very first line). As a result, if you press the Run/Stop key (ie: R/S) the program will run again. For example the following keystrokes will re-execute the same program:-



The answer is nearly 7854 miles.

#### Note



*Run/Stop is aptly named. If a program is running, this instruction will stop execution (perhaps awaiting the user entering new data). If no program is running, then this key will cause execution to begin from the address of the step in the current program pointer (see next section) and that means if the program pointer changes – the R/S key used in the example above may not work.*

## ***Program Pointer***

The calculator uses the program pointer to address one specific line in program memory, and it can be thought of as a permanent record which only changes when programs are written or updated, when they are called or indeed when you specifically change the pointers value via GTO, XEQ or the scroll keys (▼ Or ▲).

## ***Program-Entry mode***

The key sequence **h P/R** toggles the calculator in and out of program-entry mode. In this mode, functions and numbers you type on the keyboard are saved as program instructions and are not immediately executed.

## ***Program-Step Number***

While working through the example above – each new line entered caused the step number to increment. In effect, step number n points to a particular program line, and you can use the scroll keys (▼▲) to change the current step on display any time.

## ***Inserting new program steps***

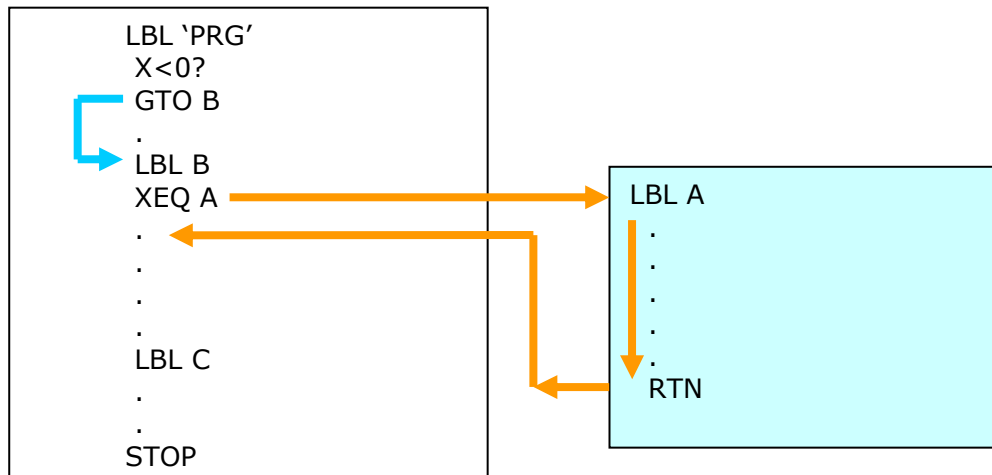
Instructions keyed into a program are inserted immediately after the current program line at which point the step number increments by one. Therefore to insert a new line between steps 1 and 2, you would position the step number on step 1, and then key in the new instruction.

## ***Deleting existing program steps***

To delete a program line, position the step number on the line to delete, and then press the backspace key (←). When you delete a line the step number moves to the previous line

## Branching and Calling – what's the difference?

Programs can invoke each other in two different ways. One program can branch to another program using the GTO instruction (often known as a jump). Alternatively one program can call another program using the XEQ instruction. Both methods are initially quite similar but the big difference with the call mechanism is that when the 2<sup>nd</sup> program executes a RTN (ie: return) instruction, execution will then return to the line immediately following the original XEQ



In this example the program 'PRG' uses a (conditional) branch (or jump) to invoke local label B (via the blue path). It then uses the XEQ instruction to "call" the subroutine with label A (via the orange path). Following the execution of the return instruction (RTN) at the end of subroutine A execution then reverts to the instruction directly following the original call.

## Program Labels

In the example circle circumference calculator program above, we used the global label 'CIR' to define the start of the program. Global labels define a program uniquely in the entire program space and when we tell the calculator to execute a global program name, the calculator searches all of the available program space to try to locate that program name. Generally a program should always start with a global label.

Within a program, individual routines can be identified with local labels.

We will look at both global and local labels in more detail next.

## Global Labels

Global labels use either 1, 2 or 3 alpha characters in a string and are distinguished by having single quote marks around the label name. An example is our example program name 'CIR'. Global names are case sensitive so 'CiR' is not the same as 'CIR' and it should be noted that the global names 'A', 'B', 'C' and 'D' are not the same as the special local hotkey names A, B, C or D (see predefined hot keys below). In effect these would be 8 distinct labels.

### Global Labels:-

1. Can be accessed no matter where the program pointer is located. The search always starts at the beginning of program memory.
2. Will be listed in the RAM catalogue (see the secondary function CAT on the STO key).
3. Should always be unique in the calculator memory to avoid confusing one program with another. (If two identical global names were to exist in program memory, then the first program in memory at the lowest step number would always be invoked first).

#### Note



*To type the letters for a global label for example after pressing XEQ, or GTO or when writing a new line of a program press ENTER to switch to alpha mode (note the single open quote that appears), and optionally press ENTER to end if using fewer than 3 characters for the label – refer to the CIR example above.*

## Local Labels

There are two types of local labels, numeric and hotkey alpha labels.

1. Numeric labels are identified by two digits 00 to 99. They can be entered directly as a two digit numeric value or can be obtained indirectly from the contents of a register. These codes are also mapped to keys on the keyboard using key coded local labels (see below)
2. There are also a set of four predefined hotkey local labels assigned to the A, B, C and D keys on the top of the calculator.

Local labels can be used to mark and provide access to various parts of a program – and they facilitate program branching.

Local labels can be...

1. Accessed ONLY within the current program (ie: the program identified by the program pointer)
2. Duplicated in separate programs – that is to say that local labels do not need to be unique within the calculator memory, but they must be unique within the confines of one single program.

## Modifying 'CIR' to demonstrate global AND hotkey local labels

To demonstrate some of the ideas involved with the two different types of labels, we will modify our CIR circumference calculation program, so that it now traps a condition where the X register contains a negative value on entry and if found will make the X register value positive before the calculation proceeds.

#### Note



*This would be dealt with more elegantly by using the abs function |x| but it suits our demonstration better to do this with a test followed by a conditional branch.*

Note that when a program encounters a test instruction such as ( $X < 0$ ) the test is evaluated and the instruction on the next line executed only if the test outcome is true. Otherwise the next instruction is skipped. Our modified program (with the test) would now look as follows:-

LBL 'CIR'
$x \geq 0?$
GTO A
-1
X
LBL A
$\pi$
X
2
X
STOP

In this particular demonstration we will use the A hotkey out of the four (A, B, C and D) available on the top row of the calculator as our local label.

The following keystrokes will be required to modify the existing CIR program

First – use GTO 'CIR' to locate the start of our circumference program which uses the global label 'CIR'. Next we switch the calculator from run into program mode using the P/R key. Note that following this key sequence, the calculator display shows the very first step of the program.

**h** **GTO** **ENTER** **C** **I** **R** **h** **P/R**

LBL'CIR' BEG STO 360  
STEP 001 5:19

Remember that inserting a new instruction when sitting on step n, will actually place the instruction at step n+1. While on step 1, insert a test to find out if the X register is  $\geq 0$ . The test is located in the TEST catalogue. The following keystrokes will open the catalogue at which point we type X to find all commands starting with the letter X. Then scroll ( $\blacktriangledown \blacktriangle$ ) until you find  $x \geq ?$  then press enter to select it. The calculator will expect an argument which can be 0 or 1 in the case of an immediate test (other values such as register numbers could be used here). We use zero.

Note that when a program encounters a test instruction such as ( $X \geq 0$ ) then the test is evaluated and the instruction on the next line executed only if the test outcome is true. We will place the branch after the test – making it conditional

**h** **TEST** **X**  **$X \geq ?$**  **ENTER** **0**

$x \geq 0?$  STO 360  
STEP 002 5:18

If the test outcome is TRUE (which will occur if the X value is positive), then this next line will execute. If it does, the calculator will branch straight to the circumference calculation code (skipping multiplication by -1). Note that for the label we use one of the four hotkeys on the top line of the calculator namely: A.

**h** **GTO** **A**

GTO A STO 360  
STEP 003 5:17

If the X register value is negative, then we will execute this part of the code designed to multiply the users X register by -1. Note you could amend this code to simply use change sign – try that if you wish to experiment.

<b>I</b>	1 STEP 004 STO 360 S 16
<b>+/-</b>	+/- STEP 005 STO 360 S 15
<b>x</b>	x STEP 006 STO 360 S 14
Now insert a new local label. We will use the A hotkey at the top of the keyboard	
<b>f</b> <b>LBL</b> <b>A</b>	LBL A STEP 007 STO 360 S 13
Feel free now that the code changes are complete to use the scroll down key to check what the next line on step 8 is. Note that it should be showing the constant Pi (constants are always prefixed by a hash) and the rest of our CIR program ie: "x", "2", "x", "END" on steps 9, 10, 11 and 12	
<b>▼</b>	# $\pi$ STEP 008 STO 360 S 13

When you're ready, exit the program

Leave program mode.	
<b>EXIT</b>	

If you now execute the modified program passing either 40 or -40, the program will check the input and adjust to ensure that the function always gives a positive result.

Therefore either this command sequence

**40** **+/-** **XEQ** **ENTER** **C** **I** **R**

or this command sequence

**40** **XEQ** **ENTER** **C** **I** **R**

will result in a display showing

BEG 360 RPN  
25.13274

## Predefined local program labels A, B, C, D – hotkey labels

In the modified CIR program above, we used the global label 'CIR' and the hotkey local label A. The calculator includes a set of four predefined hotkey local labels and which are labelled A, B, C and D. Any of these four keys can be used (and in any order) as local program labels.

*Note that these four local labels are NOT the same as global labels 'A', 'B', 'C' or 'D'*



If any of the four keys are NOT being used as a local label in the program identified by the current program pointer, then each will automatically reassign themselves a set of default and useful functions – defined for each key as follows:-

Key	Default key function when NOT assigned as a local program label
A	Sigma (cumulative stats data addition function) - shown as $\Sigma+$
B	Reciprocal - shown as $1/x$
C	Power – shown as $y^x$
D	Square root – shown as $\sqrt{x}$

### Note



*The hotkey auto assignment feature warrants a little care. Don't be surprised for example if the program pointer is pointing somewhere unexpected, and you press A expecting it to resolve to a program label, only to find the number 1 appears in the X register.*

*This scenario would occur if A actually wasn't used in the program currently addressed by the program pointer. As such, the auto assignment feature would mean that the A key would be mapped to the  $\Sigma+$  function and so pressing that key would be treated as the 1<sup>st</sup> sigma data point.*

## Key Code Labels

In addition to the four hotkey local labels A to D, it is possible to use virtually any key on the keyboard as a local label in a program. Instead of using a name, these keys are referred to in the program using a numeric key code (one per key which is based on the keys position row and column numbers on the front of the calculator). We invoke these type of labels using the execute command key (XEQ) followed by the chosen key. Internally – the calculator converts the pressed key to its key code number, and then calls the program label with that number.

One key that cannot be used in this way is the f shift (yellow) key.

In the table below, the f shift key is shown in white as it cannot be used as a key coded label, but all the others can. Note the number below each key (in brackets) is the key code for that key. For example the digit 8 key is on row 4 and position 3 from the left – hence the key code is 43.

The green keys in this table can be used directly (as in the example we will go through further down), but the other orange keys can only be used when prefixed with the yellow f shift key. That's because unless we qualify these keys (using the shift) then the calculator would interpret their meaning in a different way.

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)	X (55)	
▼ (61)	1 (62)	2 (63)	3 (64)	- (65)	
EXIT (71)	0 (72)	. (73)	R/S (74)	+ (75)	

Figure 5 – Keyboard key codes for all keys

As an example we will now open our modified circumference program at the start, delete the previously used local label A and replace both it and the call to it with a new label using the key coded "CPX" key.

In order to locate the start of the program, first use GTO (a secondary function of the XEQ key) followed by the current label – which is still currently the global label 'CIR', and then set the calculator into program mode. At that point the display will be showing the first line of the CIR program.









Refer to the following table to change the original hot key label A to a new key coded label linked to the CPX button

First – use GTO 'CIR' to locate the start of our modified circumference program which is still using an internal local label A. Next we switch the calculator from run into program mode using the P/R key. Note that following this key sequence, the calculator display shows the very first step of the program.

h GTO ENTER C I R h P/R	LBL'CIR' STEP 001 5 13
Scroll down to the branch using the scroll keys (▼▲) to locate the GTO instruction on step 3. This is the line that branches to the circumference calculation. Delete step 3	
←	x≥0? STEP 002 5 14

Now insert a new GTO instruction that this time branches to a key coded label using the CPX key. The key code will be 16 because the CPX key sits on row 1, column 6 of the keyboard. Note the way the GTO instruction label is shown simply as the numeric label 16



  	GTO 16 STEP 003 STO 360 5 13
Now scroll a little further down to get to the original definition for label A on step 7. This will be replaced with our new label for the key coded CPX key. Once you get to step 7, delete that line...	
	* STEP 006 STO 360 5 14
Now insert the new local label using the key coded CPX key	
  	LBL 16 STEP 007 STO 360 5 13
Leave program mode.	
	

With this modification in place, program execution won't change at all. The only change will be that the program will now employ a key coded local label. We execute the modified program passing either 40 or -40 and the program will again check the input and adjust to ensure that the function always gives a positive result.

Or the second key sequence

...and observe that both results are 25.132774

#### Note



*In this example - the key coded key label uses the code for the CPX key, and so in effect this label is just a numeric label with the value 16. Remember this as you read the number labels section below and in particular as you read the notes regarding direct and indirect execution (below)*

## Number labels – 00 to 99

In addition to the four hotkey local labels A to D, and key coded local labels, we can also use numeric local labels instead. These use a fixed 2 digit format, and range from 00 to 99 using decimal notation. Let us re-edit the same program in much the same way as above and change the label from the key coded CPX key, to the numeric label 32

First – use GTO 'CIR' to locate the start of our modified circumference program which is still using an internal local label A. Next we switch the calculator from run into program mode using the P/R key. Note that following this key sequence, the calculator display shows the very first step of the program.

<b>h</b> <b>GTO</b> <b>ENTER</b> <b>C</b> <b>I</b> <b>R</b> <b>h</b> <b>P/R</b>	LBL'CIR' <span style="float:right">BEG STO 360</span> STEP 001 <span style="float:right">S 13</span>
---	---

Scroll down to the branch (the GTO instruction) on step 3. This is the line that branches to the circumference calculation. Delete step 3

<b>←</b>	x≥0? <span style="float:right">STO 360</span> STEP 002 <span style="float:right">S 14</span>
----------	---

Now insert a new GTO instruction that this time branches to the numeric code 32. Note the way the GTO instruction label is again (just as per the keycoded label) shown simply as the numeric label 32

<b>h</b> <b>GTO</b> <b>32</b>	GTO 32 <span style="float:right">STO 360</span> STEP 003 <span style="float:right">S 13</span>
-------------------------------	---

Now scroll a little further down to get to the original definition for label 16 on step 7. This will be replaced with our new label for the numeric code 32. Once you get to step 7, delete that line...

<b>←</b>	* <span style="float:right">STO 360</span> STEP 006 <span style="float:right">S 14</span>
----------	--

Now insert the new local label using the numeric value 32

<b>f</b> <b>LBL</b> <b>32</b>	LBL 32 <span style="float:right">STO 360</span> STEP 007 <span style="float:right">S 13</span>
-------------------------------	---

Leave program mode.

<b>EXIT</b>	
-------------	--

With this modification in place, program execution won't change at all. The only change will be that the program will now employ a numeric local label. We execute the modified program passing either 40 or -40 and the program will check the input and adjust to ensure that the function always gives a positive result.

<b>40</b> <b>+/-</b> <b>XEQ</b> <b>ENTER</b> <b>C</b> <b>I</b> <b>R</b>
---

Or the second key sequence



and observe that both results are 25.13274

### Direct program execution using number labels

Regardless of how a local label has been created, either as a number label, or as a key coded label – the calculator treats both these labels the same way. You can for example branch to some numeric label n using the line GTO n, or you can call the numeric label n using the line XEQ n. This is known as direct execution.

Let's create a new and very simple program– using local labels throughout. This example will merely take the X register value and multiply it by 2.

Instruct the calculator to find the next free memory area for a program, and then switch into program mode.	
	<div>END <span>STO 360</span></div> <div>STEP 012 <span>513</span></div>
Now create a new local label called 32	
	<div>LBL 32 <span>STO 360</span></div> <div>STEP 013 <span>511</span></div>
Now enter 2 and multiply the result	
	<div>2 <span>STO 360</span></div> <div>STEP 014 <span>510</span></div>
	<div>x <span>STO 360</span></div> <div>STEP 015 <span>509</span></div>
	<div><span>360 RPN</span></div> <div>00000</div>

#### Note



Remember that when we exit program mode, the program pointer will still be pointing at our program. Hence we will be able to invoke the local label 32.

Use this key sequence to take a value (64) and double it



**Note**

*Incidentally – if you refer to the key code chart above, you will find that numeric label 32 happens to be the code for the X exchange Y key. As the calculator treats key code labels, and numeric labels the same way – you are perfectly at liberty to invoke the new test program using the key sequence.*

64

XEQ

X↔Y

### ***Indirect program execution using number labels***

Number labels can be executed in a slightly more interesting way using indirect addressing. This method of executing a program involves storing the number of the label linked to that program in some register p, and then instructing the calculator to indirectly execute the label number held in register p. An example will help make this clear – based on the circumference program (above) using label 32

Pick a register – let's say register 97. This will be our indirect register – and in it we will store the label of our new doubling program (ie: 32). Use the following keystrokes to store the value

32

STO

97

Now, execute the program indirectly – with 64 in the X register, using the following key strokes.

This key is located on the top row of the calculator, and when pressed in this context tells the WP 34S that an indirect input is being given. The calculator will then operate in temporary alpha mode where numbers are valid, AND SO ARE the 12 named stack registers X,Y,Z,T,A,B,C,D,I,L,J and K. Note that register X is invoked by pressing the dot key twice.

64

XEQ

→

97

The program will execute, and the display will show 128.0000

The same indirect addressing technique can also be used with any RPN stack register. For example – if we wrote the programs label into the Y register, we could then indirectly invoke the program using the Y register.

You could tackle this in one of two ways. For example you could first load the Y register with the program label using the following key strokes

**32** **STO** **ENTER** **Y**

and then indirectly invoke the program number label stored in Y

**64** **XEQ** **→** **Y**

The program will execute, and the display will show 128.0000

Another method would be to simply push the label (32) onto the stack, then push the value that is to be doubled (64) onto the stack which leaves both registers X and Y ready. You then invoke the indirect call.

**32** **ENTER** **64** **XEQ** **→** **Y**

The program will execute, and the display will show 128.0000

### ***Indirect addressing – used to access the stack***

Another feature of indirect addressing is its ability to access the stack. The 12 stack registers namely X,Y,Z,T along with A,B,C,D, and I,L, J and K all have a register number assigned, which starts from 100 for register X. Keep in mind that from the calculators point of view, any register with a number greater than 99 can only be accessed numerically using indirect addressing.

Stack register	Equivalent Register Number
X	100
Y	101
Z	102
T	103
A	104
B	105
C	106
D	107
L	108
I	109
J	110
K	111

For example – if the value 103 was stored in register 23, then the following key sequence would read the T (top of stack) register.

**RCL** **→** **23**

### ***Deleting a specific program***

In order to delete a program completely from memory – two steps are required. First – you tell the calculator to GTO the start of the program, and second you execute the CLP command (which is a secondary function on the back space key).

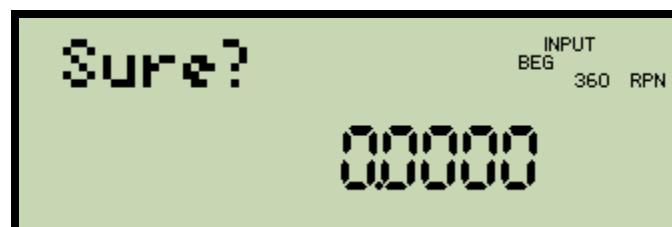
To delete our doubling test program above we would use the following key strokes. First set the calculator so that the program pointer is pointing at the start



Then delete that program



The calculator will then challenge you with an "Are you sure?" question – as follows:-



The action will proceed if, and only if the user presses the "Y" key (hits the R/S key). Otherwise the deletion will be cancelled.

### ***Deleting all programs***

To delete all programs stored in the calculator – invoke the CLPALL function in the P.FCN catalogue. Just as above, that function will challenge the user – and if the "Y" key is pressed, all programs stored on the calculators RAM will be cleared.

### ***Saving Programs to the Flash RAM Library - advanced***

Although beyond the scope of this guide, it is worth mentioning two extra commands in the programmers function catalogue that can be used to either save the current program (identified by the program pointer) to the calculators flash RAM memory area known as the library. The command is called PSTO. The program should start (ideally) with a global (alphanumeric) label and note that this operation will overwrite a previously saved program in the library if it has the same name.

You use the complementary command PRCL to copy the current program from the library back into program memory space where it can be edited (note that duplicate names are allowed) with this operation.

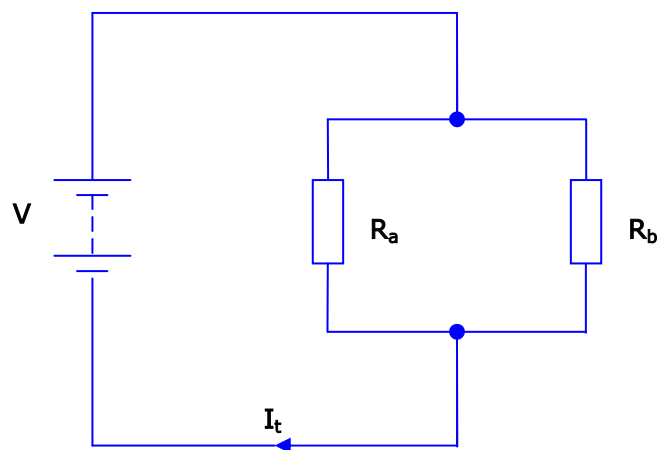
You can browse library function(s) using the CAT catalogue.

## Program Input / Output

The example circumference program (see above) was rather simple. It only had one input (the radius) and it didn't generate any kind of text prompt for that input, nor did it provide any text to explain the output. Consequently, if a user revisited the program in 6 months time, it is likely that, at best its use would be somewhat ambiguous.

The WP 34S has a handy little function useful for electronic engineers and located as a secondary function on the divide key (it is shown as two vertical parallel lines). It calculates the combined resistance of two individual resistors placed in parallel in an electrical circuit.

Consider the following electrical circuit



The combined resistance of  $R_a$  and  $R_b$  is given by the equation

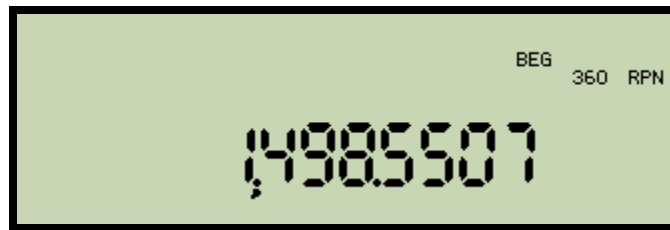
$$R_t = \frac{R_a \cdot R_b}{R_a + R_b}$$

For example - if  $R_a$  was 2K2 (ie: 2200Ω) and  $R_b$  was 4K7 (ie: 4700Ω) then the total resistance would be 1498.5507Ω. We can check the function built into the WP 34S by entering the two resistor values into the X and Y registers of the stack, and then executing the function.

Using the following keystrokes...



As expected, the calculator confirms the total resistance as:-

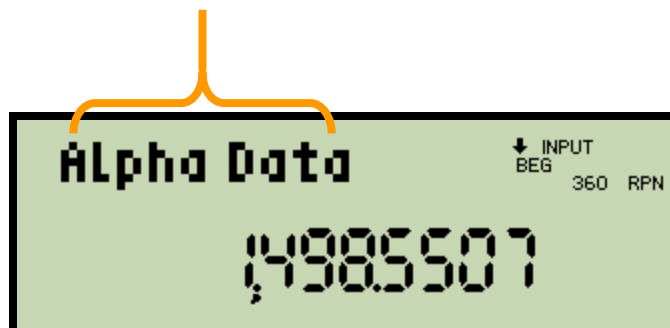


In the next program example, we are going to create a new program capable of prompting for the two resistor inputs Ra and Rb, and which will also provide a text prompt for the output (Rt) after it runs.

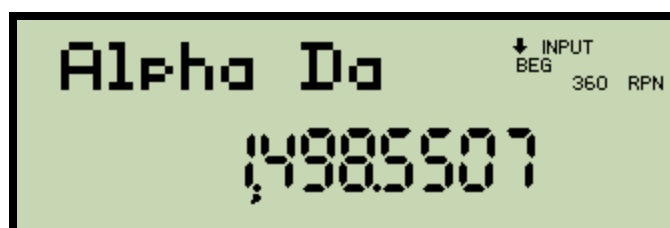
In order to do this we need to make use of the Alpha Register.

### ***The Alpha Register – how to display text***

The Alpha Register is a specially built register capable of holding up to a maximum of 31 different alpha characters. The calculators LCD display is capable of showing the alpha register, albeit on a relatively small area of the screen along with numeric data (and the normal assorted annunciators)



The processing tasked with displaying alpha data is elegant, in the sense that it reduces the size of the font only when there are sufficient characters to warrant it. For example if the final 'a' and 't' characters are deleted from the string "Alpha Data" above, then the calculator will automatically increase the character font size – as can be seen below

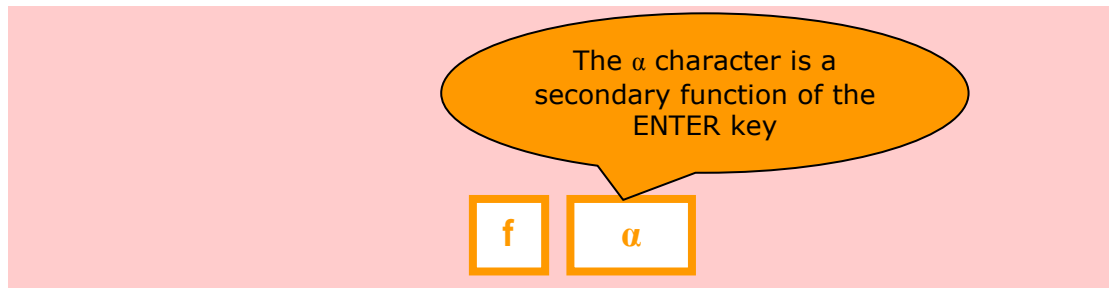


The alpha register can be cleared, have new data written into it – and more importantly, it can be switched on and off under program control. In other words,

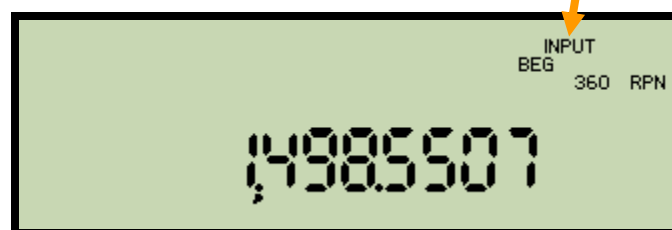


it can be used to prompt for inputs. There are a range of functions available to control the alpha register.

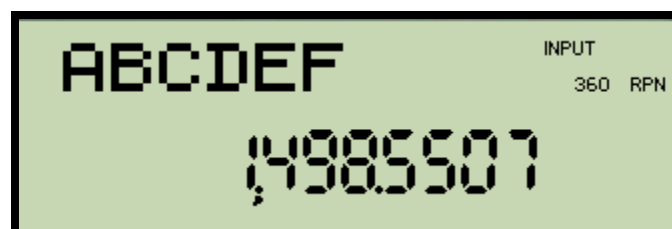
You can write immediate data into the alpha register simply by switching into the alpha input mode via the following key strokes



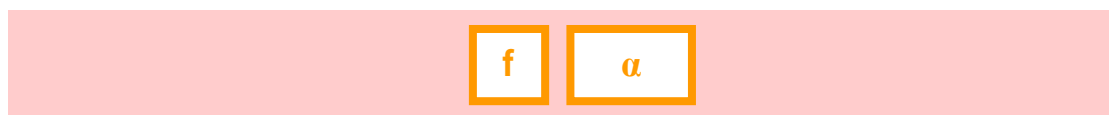
As soon as you do this, the calculator will show the "INPUT" annunciator on the top right of the display...



...and the calculator will switch into alpha entry mode. All letter keys are interpreted directly. You could type A, B, C, D, E and F (using the top row of keys on the calculator) and see the following display



At this point, you can terminate alpha mode – by repeating the key strokes...



...the input annunciator will disappear, and the keyboard will revert from alpha mode to normal mode.

Note two important points. First if you were to switch back to alpha mode now, the characters ABCDEF would still be present and you would still be able to see them. In other words, the alpha register holds information (until you specifically clear it) even when alpha mode is turned off. Second, unless you specifically clear the alpha register, the act of adding a new character will append it to the end of whatever is already there (ie: directly after 'F' in this case). If you were to add sufficient characters to overrun the length of the alpha register, then the characters at the start of the alpha register would be discarded as each new character was added.

### Inserting special characters into alpha

The alpha register permits the use of a wide range of special characters. Everything from Greek to math symbols. Some of the characters (such as punctuation and math symbols) are accessed in special alpha catalogues – that is to say, catalogues that can only be opened when the calculator is running in alpha mode. Other symbols (including Greek letters) are accessed directly via keys on the keyboard after one of the shift keys, and of course ordinary English letters are accessed direct from keys without any shift as we've seen above. Upper and lower case does have relevance for example when dealing with English or Greek letters (use the keystrokes **f** **↑** to toggle between upper and lower case).

**Note**



*Assume that the calculator is still in the immediate alpha mode described above so you should be able to see the "input" annunciator on the display.*

### Alpha Catalogues

When the calculator is in alpha mode, a set of dedicated catalogues are available to the user. These catalogues provide access to a range of math and punctuation character symbols for use in the alpha register. The relevant catalogue is first opened (see keystrokes below) and a particular character located using the scroll keys (**▲▼**). The character is copied to the alpha register by pressing ENTER – which also closes the catalogue. Alternatively EXIT can be pressed to cancel (and close) the catalogue operation.

### Common math symbol alpha catalogue

A set of common math symbols such as greater than, less than, square or round brackets, equals and others can be accessed in the **TEST** alpha catalogue which is opened using the following keystrokes when in alpha mode



The calculator will open a scrollable list of characters. Scroll through the list using the scroll keys (**▲▼**) and then press ENTER to place the selected character into the alpha register. The **TEST** alpha catalogue will then close. The symbols available in the **TEST** alpha catalogue are...

Alpha Mode					
<b>TEST</b> Catalogue					
Math Symbols					
	⌊			⌋	
	⌈			[	
	=			]	
	×			{	
	÷			}	

**Common math symbols super/subscripted alpha catalogue**

The WP 34S extends the math symbols available by providing an alpha catalogue holding super and subscripted variants of assorted symbols (ie: half height and at either the top or bottom of the character cell). This catalogue can be opened using the following key sequence.



The resulting alpha catalogue holds the following super and subscripted math symbols.

Alpha Mode R↑ Catalogue Super & Subscripted Symbols					
	□			◻	
	◻			◻	
	1			1	
	2			2	
	2			2	
	3			3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
	9			9	
	10			10	
	11			11	

Punctuation symbol alpha catalogue

The calculator has an alpha catalogue holding punctuation and style symbols including common currency symbols. The catalogue can be opened using the following keystrokes



This alpha catalogue contains the following symbols

Alpha Mode ./, Catalogue Punctuation Symbols									
	,			©			£		
	?			—			¥		
	■			~			©		
	;			—			⊗		
	'			‰			#		
	"			±					
	*			€					

### Math symbols (stats & complex domain) alpha catalogue

The calculator has an alpha catalogue holding math symbols linked to the complex domain and statistics. You can access this alpha catalogue by pressing **f** shift followed by **CPX** (ie: top right of the keyboard) as shown below

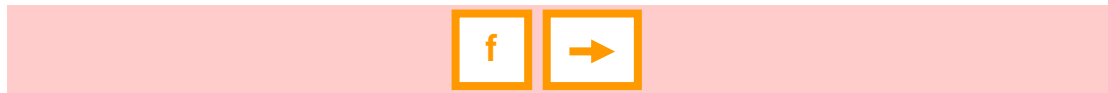


This alpha catalogue contains the following mathematical symbols linked to statistics and the complex domain (and other areas of math) – some linked to the linear estimations we will discuss in later notes.

Alpha Mode CPX Catalogue Stats and complex domain symbols											
$\sigma$	$\sigma^2$	$\sigma^3$	$\sigma^4$	$\sigma^5$	$\sigma^6$	$\sigma^7$	$\sigma^8$	$\sigma^9$	$\sigma^{10}$	$\sigma^{11}$	$\sigma^{12}$
$\sigma^{13}$	$\sigma^{14}$	$\sigma^{15}$	$\sigma^{16}$	$\sigma^{17}$	$\sigma^{18}$	$\sigma^{19}$	$\sigma^{20}$	$\sigma^{21}$	$\sigma^{22}$	$\sigma^{23}$	$\sigma^{24}$
$\sigma^{25}$	$\sigma^{26}$	$\sigma^{27}$	$\sigma^{28}$	$\sigma^{29}$	$\sigma^{30}$	$\sigma^{31}$	$\sigma^{32}$	$\sigma^{33}$	$\sigma^{34}$	$\sigma^{35}$	$\sigma^{36}$
$\sigma^{37}$	$\sigma^{38}$	$\sigma^{39}$	$\sigma^{40}$	$\sigma^{41}$	$\sigma^{42}$	$\sigma^{43}$	$\sigma^{44}$	$\sigma^{45}$	$\sigma^{46}$	$\sigma^{47}$	$\sigma^{48}$
$\sigma^{49}$	$\sigma^{50}$	$\sigma^{51}$	$\sigma^{52}$	$\sigma^{53}$	$\sigma^{54}$	$\sigma^{55}$	$\sigma^{56}$	$\sigma^{57}$	$\sigma^{58}$	$\sigma^{59}$	$\sigma^{60}$
$\sigma^{61}$	$\sigma^{62}$	$\sigma^{63}$	$\sigma^{64}$	$\sigma^{65}$	$\sigma^{66}$	$\sigma^{67}$	$\sigma^{68}$	$\sigma^{69}$	$\sigma^{70}$	$\sigma^{71}$	$\sigma^{72}$
$\sigma^{73}$	$\sigma^{74}$	$\sigma^{75}$	$\sigma^{76}$	$\sigma^{77}$	$\sigma^{78}$	$\sigma^{79}$	$\sigma^{80}$	$\sigma^{81}$	$\sigma^{82}$	$\sigma^{83}$	$\sigma^{84}$
$\sigma^{85}$	$\sigma^{86}$	$\sigma^{87}$	$\sigma^{88}$	$\sigma^{89}$	$\sigma^{90}$	$\sigma^{91}$	$\sigma^{92}$	$\sigma^{93}$	$\sigma^{94}$	$\sigma^{95}$	$\sigma^{96}$
$\sigma^{97}$	$\sigma^{98}$	$\sigma^{99}$	$\sigma^{100}$	$\sigma^{101}$	$\sigma^{102}$	$\sigma^{103}$	$\sigma^{104}$	$\sigma^{105}$	$\sigma^{106}$	$\sigma^{107}$	$\sigma^{108}$
$\sigma^{109}$	$\sigma^{110}$	$\sigma^{111}$	$\sigma^{112}$	$\sigma^{113}$	$\sigma^{114}$	$\sigma^{115}$	$\sigma^{116}$	$\sigma^{117}$	$\sigma^{118}$	$\sigma^{119}$	$\sigma^{120}$

## Arrows and extra math symbol alpha catalogue

There is an alpha catalogue holding arrow symbols and some common math symbols (such as infinity and root). This catalogue is opened using the following key sequence...



The catalogue will open to show the following symbols.

Alpha Mode → Catalogue Arrow Symbols					
	→			∞	
	←			∅	
	↑			^	
	↓			÷	
	√				

## Other alpha characters obtained direct from the keyboard

Over and above the characters available in alpha catalogues, there are a set of extra characters which are available directly from keys on the keyboard when the calculator is working in alpha mode. These characters are inserted into the alpha register as soon as the key stroke sequence is used. We look at these characters next.

## English alpha characters (A-Z, a-z)

English characters are entered directly from the keyboard by pressing the unshifted key shown with the desired character shown in grey at its bottom left. Note that all English letters can be either upper or lower case. Use the keystrokes **f** **↑** to toggle between upper and lower case.

## Number digits (0 to 9)

Numbers are inserted directly from the keyboard by **f** shifting the relevant digit key 0 to 9.

### Note



*In fact, the digits 0 and 1 are not linked to alpha characters anyway – so you actually don't need to **f** shift either of these two digits, although it doesn't matter if you do.*

## Logic symbols

When in alpha mode, the following **h** shifted key symbols can be obtained.

Key sequence	Symbol
<b>h</b> <b>!</b>	<b>!</b>
<b>h</b> <b>AND</b>	<b>&amp;</b>
<b>h</b> <b>OR</b>	<b> </b>
<b>h</b> <b>XOR</b>	<b>≠</b>
<b>h</b> <b>NOT</b>	<b>¬</b>

## Other useful **h** shifted misc symbols – including a space

Other useful symbols direct from **h** shifted keys are shown below. Note the last sequence inserts a space character into the alpha register

Key sequence	Symbol
<b>h</b> <b>π</b>	<b>π</b>
<b>h</b> <b>X▶</b>	<b>±</b>
<b>h</b> <b>RMDR</b>	<b>\</b>
<b>h</b> <b>STATUS</b>	<b>?</b>
<b>h</b> <b>PSE</b>	<b> </b>

## Other useful **f** shifted math symbols

Other useful symbols direct from **f** shifted keys are as shown below

Key sequence	Symbol
<b>f</b> <b>÷</b>	<b>/</b>
<b>f</b> <b>×</b>	<b>×</b>
<b>f</b> <b>-</b>	<b>-</b>
<b>f</b> <b>+</b>	<b>+</b>
<b>f</b> <b>X▶Y</b>	<b>()</b>
<b>f</b> <b>+/-</b>	<b>±</b>

## Greek character symbols

The calculator will allow the insertion of Greek symbols – both upper and lower case. The following table illustrates the mapping between keys on the keyboard and their corresponding Greek symbols once the key has been **g** shifted in alpha mode. Note that all Greek symbols can be either upper or lower case. Use the keystrokes **f** **↑** to toggle between upper and lower case.

Alpha Catalogue of upper and lower case Greek letters			
Key Assignment	Greek letters (in order)	Upper case	Lower Case
A	1. Alpha	Α	α
B	2. Beta	Β	β
C and/or G	3. Gamma	Γ	γ
D	4. Delta	Δ	δ
E	5. Epsilon	Ε	ε
Z	6. Zeta	Ζ	ζ
The ENTER key	7. Eta	Η	η
1 (ie: digit one)	8. Theta	Θ	θ
I	9. Iota	Ι	ι
K	10. Kappa	Κ	κ
L	11. Lambda	Λ	λ
M	12. Mu	Μ	μ
N	13. Nu	Ν	ν
X	14. Xi	Ξ	ξ
<i>Note: Omicron is omitted because the character is essentially identical to O when rendered</i>			
P	15. Pi	Π	π
R	16. Rho	Ρ	ρ
S	17. Sigma	Σ	σ
T	18. Tau	Τ	τ
Y	19. Upsilon	Υ	υ
F	20. Phi	Φ	φ
H	21. Chi	Χ	χ
0 (ie: digit zero)	22. Psi	Ψ	ψ
O	23. Omega	Ω	ω
J	Close bracket symbol	⌋	⌋
Q	(Nothing assigned to this key)		
U	(Nothing assigned to this key)		
V	(Nothing assigned to this key)		
W	(Nothing assigned to this key)		

A useful reference for the Greek alphabet is: [http://en.wikipedia.org/wiki/Greek\\_alphabet](http://en.wikipedia.org/wiki/Greek_alphabet)



## Alpha Register Commands

The following table defines some of the most useful alpha commands.

VIEW r	<p>Secondary function for the RCL key</p> <p>When a program is started, the display contents are replaced by the "Running Program" message.</p> <p>You can display a number while the program is running using the "VIEW" command. When VIEW is invoked the keyboard switches into temporary alpha mode and will then accept either a numeric register number 00 to 99, or one of the special register letters (including those of the stack registers (X,Y,Z,T,A,B,C,D,I,L,J or J)). View then updates the display (and it does so in a way designed to reduce flicker – so there is some overhead each time this command executes). As soon as the display is updated, control immediately continues with the next instruction.</p> <p>You could for example use VIEW to display the X register to the user, as the program is running.</p>
PSE n	<p>Secondary function for the digit 0 key</p> <p>The PSE command simply pauses calculator execution. PSE takes an argument n to set the pause delay in multiples of one tenth of a second. During this interval, the display will show the X register for the time delay defined by n. A delay of zero is actually the same as "VIEW X"</p> <p>If you wish to make the view display persist, simply follow it with a PSE n command</p>
Cl $\alpha$	<p>Function in the P.FCN catalogue</p> <p>This command clears the alpha register</p>
$\alpha$ View	<p>Function in the P.FCN catalogue</p> <p>This command displays the alpha register without displaying any number. The numeric portion of the display is replaced by three hyphens.</p>
$\alpha$ ON	<p>Function in the P.FCN catalogue</p> <p>This command switches the alpha register on – but comes with the disadvantage that it displays the tail of the alpha string if it is quite long. It also leaves the alpha register on so if the program terminates, the user will find themselves in alpha mode (which can be quite confusing)</p>
$\alpha$ OFF	<p>Function in the P.FCN catalogue</p> <p>Switches the alpha display off</p>
VW $\alpha$ +	<p>Function in the P.FCN catalogue</p> <p>This special command is a combination of functions. It enables the display of the alpha register, while also displaying the contents of any register (just as per view). The name can be thought of as view with alpha plus register.</p>
PROMPT	<p>Function in the P.FCN catalogue</p> <p>This command in the P.FCN catalogue, is a combination of VW<math>\alpha</math>+ X followed by STOP (to await the users input).</p>

### Note



*Regarding alpha character entry in the WP 34S when in program mode, you can enter alpha characters one at a time (one per program step) into a program. However that method is quite wasteful of program memory. An alternative method allows you to compress three alpha characters into one instruction. (A nine character string would for example only require three instructions when compressed in this way). In order to select this mode, you enter the (yellow shifted)  $\alpha$  character twice when you are in the process of typing the alpha string data into the program.*

*The next example will demonstrate this compression technique...*

## Parallel Resistor Program using static Alpha displays

This program will request the input of two resistor values (Ra & Rb). It will then calculate the combined resistance, and display the total value Rt.

Start by instructing the calculator to find the next free space for a new program, and then switch the calculator from run into program mode.

<b>h</b> <b>GTO</b> <b>.</b> <b>.</b> <b>h</b> <b>P/R</b>	END STEP 012	STO 360 513
---	-----------------	----------------

Now create a new global label called PAR. This will be the name our program will use.

<b>f</b> <b>LBL</b> <b>ENTER</b> <b>P</b> <b>A</b> <b>R</b>	LBL'PAR' STEP 013	STO 360 510
---	----------------------	----------------

Now clear the alpha register for the first of our prompts.

<b>h</b> <b>P.FCN</b> <b>CL<math>\alpha</math></b> <b>ENTER</b>	CL $\alpha$ STEP 014	STO 360 509
---	-------------------------	----------------

With the alpha register cleared, we are now going to fill the alpha register with the string "Enter Ra". We start by typing the  $\alpha$  character to switch into alpha entry mode. Note the way the display now shows the input annunciator

<b>f</b> <b><math>\alpha</math></b>	CL $\alpha$ STEP 014	INPUT STO 360 509
-------------------------------------	-------------------------	----------------------

We could now start typing one character at a time OR, we can instruct the calculator to compress the first three alpha characters we type into one instruction. We signal that we want this compression by typing the alpha sequence one more time. Note how the left hand now shows a new instruction called alpha with a start quote.

<b>f</b> <b><math>\alpha</math></b>	$\alpha'$ STEP 014	INPUT STO 360 509
-------------------------------------	-----------------------	----------------------

We can now enter the first three characters of our string ie: 'Ent'. As soon as we have entered the 3<sup>rd</sup> character, the calculator will close this instruction (which can only hold a max of three characters). To enter the next sequence of three, we repeat the process. Note to toggle between upper and lower case use **f**  $\uparrow$

<b>E</b> <b>f</b> <b><math>\uparrow</math></b> <b>n</b> <b>t</b>	$\alpha'$ 'Ent' STEP 015	INPUT STO 360 507
--	-----------------------------	----------------------







<b>f</b> <b><math>\alpha</math></b>	$\alpha'$ STEP 015	INPUT STO 360 507
-------------------------------------	-----------------------	----------------------

Note that a space character is given by the PSE function (on the 0 digit). Again, as soon as the third character is entered (ie: the space) the calculator will terminate the instruction.

<b>e</b> <b>r</b> <b>h</b> <b>PSE</b>	$\alpha'$ er ' STEP 016	INPUT STO 360 505
---------------------------------------	----------------------------	----------------------

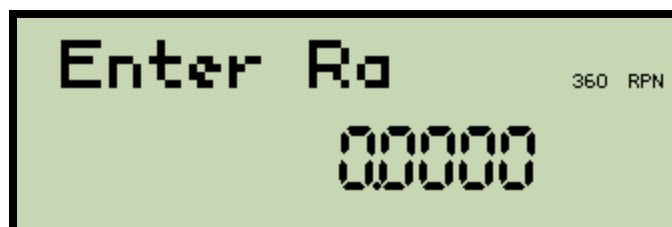
<b>f</b> <b><math>\alpha</math></b>	$\alpha'$ STEP 016	INPUT STO 360 505
-------------------------------------	-----------------------	----------------------

Enter the final two characters (capitalising the first) and then press enter to




terminate the string	
  R   a ENTER	<div> <math>\alpha</math>'Ra' <small>INPUT STO 360</small>  STEP 017 503 </div>
Then press enter one final time to terminate the alpha entry mode – note the input annunciator disappears.	
ENTER	<div> <math>\alpha</math>'Ra' <small>STO 360</small>  STEP 017 503 </div>
We can then execute the PROMPT command to display the alpha register, and the X register and wait for the user to enter data. Clear the X register before we make the call – which just ensures the user will see an X=0 display for tidiness. Don't forget to type ENTER after PROMPT (not shown for clarity)	
 CLx  P.FCN PROMPT	<div> PROMPT <small>STO 360</small>  STEP 019 501 </div>

**Note**

*If you were to exit out of program mode now, and execute this program it would display the following screen and then await the users input. It wouldn't do anything after that – other than to stop.*



We now need to add the next part of the code which prompts for Rb

Now clear the alpha register for the second of our prompts	
 P.FCN CL $\alpha$ ENTER	<div> CL<math>\alpha</math> <small>STO 360</small>  STEP 020 500 </div>
With the alpha register cleared, we are now going to fill the alpha register with the second prompt string "Enter Rb". We start by typing the $\alpha$ character to switch into alpha entry mode. Note the way the display now shows the input annunciator	
 $\alpha$	<div> CL<math>\alpha</math> <small>INPUT STO 360</small>  STEP 020 500 </div>
Next instruct the calculator to compress the first three alpha characters we type into one instruction. We signal we want that compression by typing the alpha sequence one more time. Note how the left hand now shows a new instruction called alpha with a start quote.	
 $\alpha$	<div> <math>\alpha</math>' <small>INPUT 360</small>  STEP 020 500 </div>
We can now enter the first three characters of our string ie: 'Ent'. As soon as we have entered the 3 <sup>rd</sup> character, the calculator will close this instruction (which can only hold a max of three characters). To enter the next sequence of three, we repeat the process.	

<b>E</b> <b>f</b> <b>↑</b> <b>n</b> <b>t</b>	α'Ent' <small>↑ INPUT STO 360</small> STEP 021 498
<b>f</b> <b>α</b>	α' <small>↑ INPUT 360</small> STEP 021 498
Note that a space character is given by the PSE function (on the 0 digit). Again, as soon as the third character is entered (the space) the calculator will terminate the instruction.	
<b>e</b> <b>r</b> <b>h</b> <b>PSE</b>	α'er ' <small>↑ INPUT STO 360</small> STEP 022 496
<b>f</b> <b>α</b>	α' <small>↑ INPUT 360</small> STEP 022 496
Enter the final two characters (capitalising the first) and then press enter to terminate the string	
<b>f</b> <b>↑</b> <b>R</b> <b>f</b> <b>↑</b> <b>b</b> <b>ENTER</b>	α'Rb' <small>↑ INPUT STO 360</small> STEP 023 494
Press enter to terminate the alpha entry mode. Note that the input annunciator disappears.	
<b>ENTER</b>	α'Rb' <small>STO 360</small> STEP 023 494
We can then use PROMPT to display the alpha register, and the X register and wait for the user to enter data. Note that when this runs, the X register will already hold the value of Ra. When prompt executes, the value of Ra will have been pushed onto the stack into Y, leaving X free for the users Rb entry.	
<b>h</b> <b>P.FCN</b> <b>PROMPT</b> <b>ENTER</b>	PROMPT <small>STO 360</small> STEP 024 493

We now have the two sections of code that together prompt for the inputs Ra and Rb. Once code execution reaches this point, the value of Ra will be on the Y register of the stack, and Ra will be in the X register. We can now call the || calculation function, and display the result. Let's add the lines to do that next.

Execute the parallel calculation function	
<b>g</b> <b>  </b>	<small>STO 360</small> STEP 025 492
Now clear the alpha register ready for the new text for the result prompt	
<b>h</b> <b>P.FCN</b> <b>CLα</b> <b>ENTER</b>	CLα <small>STO 360</small> STEP 026 491
With the alpha register cleared, fill the alpha register with the final result prompt string "Rt". We start by typing the α character to switch into alpha entry mode. Note the way the display now shows the input annunciator	
<b>f</b> <b>α</b>	CLα <small>INPUT STO 360</small> STEP 026 491

Next instruct the calculator to compress the alpha characters we type into one instruction. We signal we want that compression by typing the alpha sequence one more time. Note how the left hand now shows a new instruction called alpha with a start quote.

<b>f</b> <b>α</b>	α' STEP 026 491 INPUT 360
-------------------	---------------------------------

We can now enter the two characters of our string ie: 'Rt'. We terminate the 2 character string by typing enter after the last character

<b>R</b> <b>f</b> <b>↑</b> <b>t</b> <b>ENTER</b>	α'Rt' STEP 027 489 * INPUT ST0 360
--	--

Press enter to terminate the alpha entry mode. Note that the input annunciator disappears

<b>ENTER</b>	α'Rt' STEP 027 489 ST0 360
--------------	----------------------------------

With the result in X, and the alpha register set with the result prompt – we can now display the alpha, and X register and then stop

<b>h</b> <b>P.FCN</b> <b>PROMPT</b> <b>ENTER</b>	PROMPT STEP 028 488 ST0 360
--	-----------------------------------

<b>EXIT</b>	00000 360 RPN
-------------	------------------

To run the program, use the following keystrokes

<b>XEQ</b> <b>ENTER</b> <b>P</b> <b>A</b> <b>R</b>
--

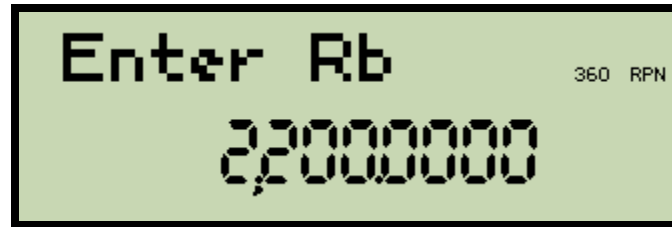
The display will show the first prompt (note that X will be clear)

Enter Ra 00000 360 RPN
------------------------------

The user would then type their first Ra resistor value – for example 2200Ω and then press the R/S key. The exact sequence would be...

<b>2200</b> <b>R/S</b>
------------------------

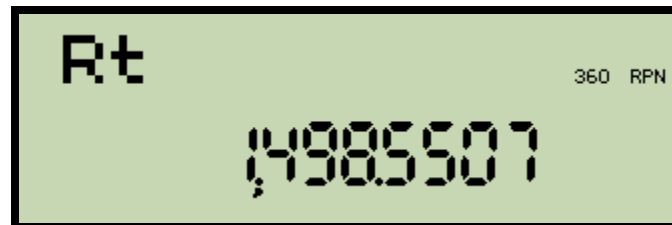
As soon as the run/stop (R/S) key is pressed, the program will continue execution at the line following the first PROMPT instruction. The display will then show...



The user would type their second Rb resistor value – for example 4700Ω and press the R/S key. The exact sequence would be...



Again - as soon as the run/stop (R/S) key is pressed, the program will continue execution at the line following the second PROMPT instruction – this time leading to the display of the final result. The display will show...

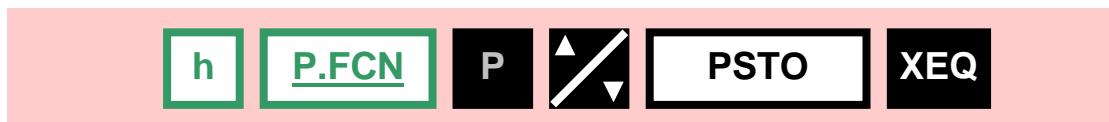


The program appears to work ok

### Archiving the program to the flash RAM library

Although beyond the scope of this manual, it is worth demonstrating how it is possible to archive the PAR parallel resistor program (entered above) into the calculator library system residing in flash RAM. To do this you use the PSTO command in the P.FCN catalogue which copies the program identified by the current value of the program pointer into the library flash RAM.

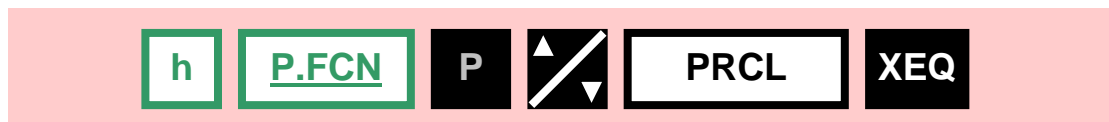
Assuming that the program pointer is still sitting on the PAR program use the following keystrokes to copy the PAR program to library flash RAM



Let's assume that after executing the keystrokes above, you then cleared all program memory using **CLP**.

If you subsequently wished to restore the library copy of the PAR program to main program memory you would use the following steps.

1. Open the **CAT** catalogue and use the UP arrow to locate the library function PAR.
2. Press ENTER to select this function in the library memory
3. Then execute the following keystrokes to restore it to program memory



After these steps are carried out, the program will once again be resident in main program memory, and can be executed and edited in the normal way.

## Debugging the parallel resistor program

Assume for a moment that the Rt value was not correct after the program had fully run and displayed its result – how might one debug the program?

This program is similar to many, in the sense that you can split the whole program into two halves. The first being the input gathering (and if you are wise this stage must include input validation) and the second half being the actual calculation. In this case, by the time you reach step 25 (where the WP 34S parallel function || is called) both the resistor values should be on the stack (Ra in stack register Y, and Rb in stack register X). A good way to verify that this is correct is to edit the program and insert a STOP just before the || function is invoked and then make sure (using the external command SHOW) that the right values are in the right place

### Note



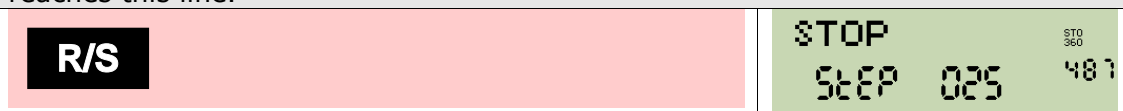
*This is known as inserting a break point*

Observe that you can use the GTO command to set the program pointer to a particular line n using the key sequence "GTO .n". We will use this initially to position for the insert.

Start by positioning the program pointer at the instruction directly before step 25 (we will insert the new command so that it preceeds the || instruction on step 25). Then switch the calculator into program mode.



Now insert a stop instruction (use R/S) to force the program to stop once it reaches this line.



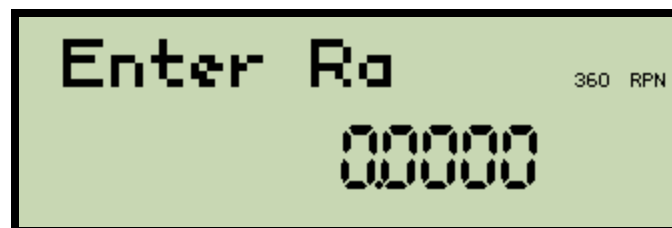
<b>EXIT</b>	
-------------	--

Before you exit you can optionally use the scroll keys (**▲▼**) to make sure that the order of the commands after the new insert is PROMPT, STOP followed by ||

To run this debug version of the program, use the following keystrokes

<b>XEQ</b>	<b>ENTER</b>	<b>P</b>	<b>A</b>	<b>R</b>	
------------	--------------	----------	----------	----------	--

The display will show the first prompt

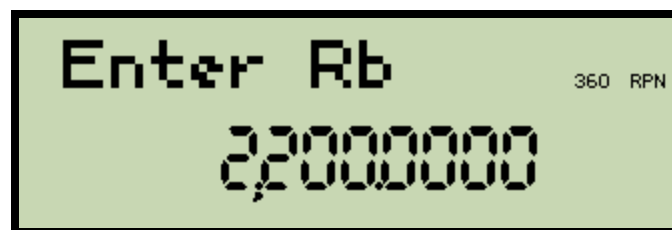


The image shows a green LCD screen with a black border. The text 'Enter Ra' is displayed in a large, pixelated font on the top line. Below it, the number '00000' is displayed in the same font. In the top right corner, the text '360 RPN' is visible in a smaller font.

The user would then type their first Ra resistor value – for example 2200Ω and then press the R/S key. The exact sequence would be...

<b>2200</b>	<b>R/S</b>
-------------	------------

As soon as the run/stop (R/S) key is pressed, the program will continue execution at the line following the first PROMPT instruction. The display will then show...



The image shows a green LCD screen with a black border. The text 'Enter Rb' is displayed in a large, pixelated font on the top line. Below it, the number '22000000' is displayed in the same font. In the top right corner, the text '360 RPN' is visible in a smaller font.

The user would then type their second Rb resistor value – for example 4700Ω and then press the R/S key. The exact sequence would be...

<b>4700</b>	<b>R/S</b>
-------------	------------

At that point the program will encounter the STOP instruction and the program will stop. The display will simply show the current contents of the X register ie:





However, at this breakpoint, we can now invoke the SHOW command (a secondary function on the EXIT key) in order to inspect the contents of the stack (and in fact any register).

The following sequence invokes the show command which will immediately display the X register.



Use the scroll down key to show the next stack register in the set (ie: Y)



And keep scrolling if you want to in order to inspect all 12 stack and special registers. Armed with this debug information – you would almost certainly be able to solve any problem with the programs operation

## Simple Counting Program – showing a dynamic Alpha display

A very simple counting program – which illustrates the use of a dynamic alpha display, would work as follows:-

LBL 'LP1'
Setup alpha
LBL A
1
+
Pause
GTO A

This program takes whatever is in the X register, adds one to it, pauses and then loops back to the start. The pause is what we will explore here. A simple alpha command called  $VW_{\alpha+}$  can be used to allow us to see the contents of both the alpha register and the X register. Execution continues immediate after the display has been updated. We will set the alpha register to show the word "COUNTING" before the program runs.

Start by instructing the calculator to find the next free space for a new program, and then switch the calculator from run into program mode.

<b>h</b> <b>GTO</b> <b>.</b> <b>.</b> <b>h</b> <b>P/R</b>	END STEP 029 487
---	---------------------

Now create a new global label called LP1. This will be the name our program will use. Note that the digits 0 and 1 are two special cases when running in alpha mode – which are obtained by f shifting the digit 1 key

<b>f</b> <b>LBL</b> <b>ENTER</b> <b>L</b> <b>P</b> <b>f</b> <b>1</b>	LBL'LP1' STEP 030 485
--	--------------------------

Now clear the alpha register for the first of our prompts.

<b>h</b> <b>P.FCN</b> <b>CL<math>\alpha</math></b> <b>ENTER</b>	CL $\alpha$ STEP 031 484
---	-----------------------------

With the alpha register cleared, we are now going to fill the alpha register with the string "COUNTING". We start by typing the  $\alpha$  character to switch into alpha entry mode. Note the way the display now shows the input annunciator

<b>f</b> <b><math>\alpha</math></b>	CL $\alpha$ STEP 031 484
-------------------------------------	-----------------------------

Next instruct the calculator to compress the first three alpha characters we type into one instruction (ie: COU). We signal we want that compression by typing the alpha sequence one more time. Note how the left hand now shows a new instruction called alpha with a start quote.

<b>f</b> <b><math>\alpha</math></b>	$\alpha'$ STEP 031 484
<b>C</b> <b>O</b> <b>U</b>	$\alpha'$ COU' STEP 032 482

<b>f</b> <b><math>\alpha</math></b>	$\alpha'$ STEP 032 INPUT 360 482
<b>N</b> <b>T</b> <b>I</b>	$\alpha'$ 'NTI' STEP 033 INPUT STO 360 480
<b>f</b> <b><math>\alpha</math></b>	$\alpha'$ STEP 033 INPUT 360 480
<b>N</b> <b>G</b> <b>ENTER</b>	$\alpha'$ 'NG' STEP 034 INPUT STO 360 478
Note that the "input" annunciator is still live – which means we are off course still in alpha mode. Hit the ENTER key to conclude that mode now.	
<b>ENTER</b>	$\alpha'$ 'NG' STEP 034 STO 360 478
Now that the alpha register is setup with its fixed string "COUNTING", we need to define a label (which we can loop to) and add the simple math. Note we will use one of the local hotkey labels A.	
<b>f</b> <b>LBL</b> <b>A</b>	LBL A STEP 035 STO 360 477
<b>1</b>	1 STEP 036 STO 360 476
<b>+</b>	1 STEP 036 STO 360 476
<b>+</b>	1 STEP 036 STO 360 476
Now insert the VW $\alpha$ + command by selecting it in the catalogue and then hitting ENTER. VW $\alpha$ + will then prompt for an argument that can be numeric (for a register value) or it can use temporary alpha mode to accept a letter designated as one of the 12 stack registers namely X,Y,Z,T,A,B,C,D,I,L,J and K. This argument defines what numeric value is seen.	
In this case we want to view the X register so in order to switch VW $\alpha$ + into temporary alpha mode we press the ENTER key a second time at which point the display will show a small "s" to signify special registers.	
After that we type the X key to signify that we want to view the X register. This is broken this down into two steps so you can see the intermediate display.	
<b>h</b> <b>P.FCN</b> <b>VW<math>\alpha</math>+</b> <b>ENTER</b> <b>ENTER</b>	VW $\alpha$ + s_ STEP 037 360 474
Note this is the letter X key (not the multiply key) – signifying stack register X.	
<b>X</b>	VW $\alpha$ + X STEP 038 STO 360 474
With the X register now momentarily displayed – loop back to the math part of the loop defined by local label A	

h GTO A	GTO A STEP 039 STO 360 473
EXIT	

Use the following key strokes to execute the counting program



**Note**



1. Hitting the EXIT key at any time will terminate the program
2. Running this on the PC emulator will run, but the display will be updated so quickly it will be virtually unreadable. A physical calculator is slow enough so that the display is clearly visible.

## Counting Program illustrating Loop Control

Even though this counting program is crude, it serves to demonstrate the principal of looping – albeit indefinitely. We could modify this program to loop a fixed number of times using one of the set of loop control instructions. A suitable example is the Decrement and Skip command known as “DSE”. This command has an argument *r* linked to it and which identifies a register. The register is filled with a value used to define the loops starting / current value, its stopping value, and the delta value applied to the counter after each iteration. The contents of register *r* has the following format

Reg *r* = CCCCCC.ffffii

Assuming register *r* contains a number in this format, then when DSE executes it decrements *r* by integer *ii*, and skips the next instruction if *cccccc* ≤ *fff*. Note that if register *r* doesn’t contain a fractional part, then the function defaults to *fff*=0, and defaults to *ii*=1. Obviously neither *fff* nor *ii* can be negative and so DSE only makes sense when *cccccc* > 0

We will modify our counting program by firstly writing a sensible starting value into some register, and then adding the DSE instruction before the jump on the last line as follows

LBL 'LP1'
Setup alpha
LBL A
1
+
Pause
DSE r
GTO A
STOP

When the register r reaches it stop condition – the GTO A instruction will be skipped, and the program will stop. We will (randomly) use register 5 as our loop control register.

The programs GTO instruction is sitting on step 39. Given we want to insert the new DSE instruction directly before that instruction, use GTO to position the program pointer to step 38 before we switch into program mode.

Position the program pointer and then switch the calculator from run into program mode.	
<b>h</b> <b>GTO</b> <b>.</b> <b>38</b> <b>h</b> <b>P/R</b>	VWα+ X STEP 038 STO 360 472
<b>f</b> <b>DSE</b> <b>05</b>	DSE 05 STEP 039 STO 360 472
<b>EXIT</b>	

We will allow the loop to count from 999 to 0, in increments of 1 – which means ccccc=999, fff=0 and ii=1 – giving the register contents of 999.00001. So before we execute the modified program write the counter value into register 5

<b>99900001</b> <b>STO</b> <b>05</b>
--------------------------------------

Now run the loop controlled program clearing the X register before we start (so that the final count value in X makes some degree of sense).

Keep in mind that unlike our first example counting program this modified program **will** exit after 999 loops. That means that the alpha register will remain on when the program exists. Note that the P.FCN catalogue command αOFF will switch the alpha display off whenever you require.

<b>0</b> <b>XEQ</b> <b>ENTER</b> <b>L</b> <b>P</b> <b>f</b> <b> </b>
--

One the program is finished, the resulting display will be:

<b>COUNTING</b>	360 RPN
<b>9990000</b>	

## Local Data - Variables

Those readers with programming experience will be familiar with the expression "local variable". The idea being that it is useful for a function to have access to a local variable or variables which remain in scope during a functions execution and which go out of scope (are lost) when the function terminates

The WP 34S offers something similar although the implementation it uses is rather different. It is also a little tricky to use safely given the memory involved is drawn from the subroutine return stack which is subject to dynamic change. There are three functions involved, and one special form of notation.

### Note



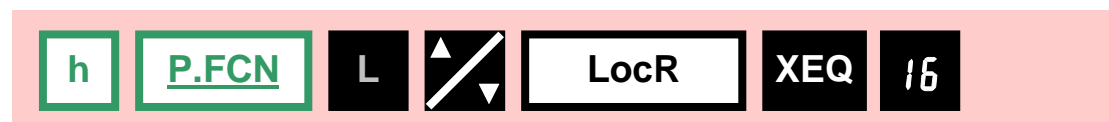
*Although the following functions can all be used fully from the keyboard, the implementation masks the fact that these functions can and will fail when apparently unrelated events take place (an example below will make this clear).*

Let us first look at the three functions involved

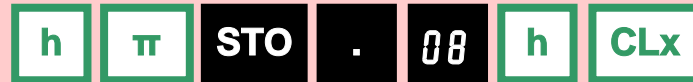
1. "LocR" n (found in the P.FCN catalogue) is the function used to allocate n local data registers and a fixed set of 16 Boolean flags. If the argument n is 1 to 16, then the resulting registers can all be directly accessed by STO or RCL using the special dot notation .00 to .15. The flags are always accessed using the dot notation .00 to .15. Note that "LocR" will allow (under specific conditions) an allocation of up to 144 registers, but the registers above the first 16 can only be accessed indirectly – and at the time of writing the mechanism involved is far from clear. All the following examples confine themselves to allocating a maximum of 16 local data registers.
2. "PopLR" (found in the P.FCN catalogue) is the function used to de-allocate any and all previously allocated local register data. The PopLR function takes no arguments.
3. "LocR?" (found in the TEST catalogue) allows the user to query the current number of allocated local data registers

When the LocR function is executed (either immediately on the keyboard, or within the body of a program) the required number of registers are pushed onto the subroutine return stack. So long as the stack isn't altered, the user will be able to use the special dot notation to access any of these registers. As soon as the PopLR function is executed the memory linked to these data registers will be cleared.

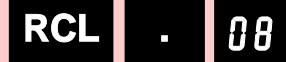
Let us now try a worked example. First let's allocate a total of 16 local data registers using the LocR command in the P.FCN catalogue



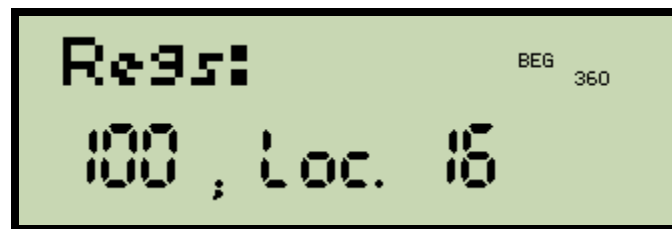
With the 16 local data registers allocated, let us now write the value of Pi into local data register 8 (remember we use the special dot notation to access these registers using .00 to .15). After this we will clear the X register – as follows:-



Next read back the local data register 8



And the display will show the value of Pi. Observe that if you were to now open the STATUS command (see above examples), and scroll down to the 2<sup>nd</sup> page (press the down scroll key one time), you would see the number of allocated local data registers prefixed by the word "Loc"



To clear the allocation of any/all local data, simply execute the "PopLR" function



### Local Data - Flags

Whenever local data is successfully allocated with the LocR command (regardless of how much) the system will automatically create a fixed set of 16 Boolean flags which can each be set, cleared and tested (see command FC? which is used to test if a flag is clear or the three commands FC?C, FC?F and FC?S which are used to clear, flip or set the flag directly after the test is performed. Similarly the FS? command is used to test if a flag is set and the three commands FS?C, FS?F, FS?S are used to clear, flip or set the flag directly after the test is performed).

Flag addressing is the same as when addressing local data variables using the same dot notation (.00 to .15). For example to set flag 8 we would use.

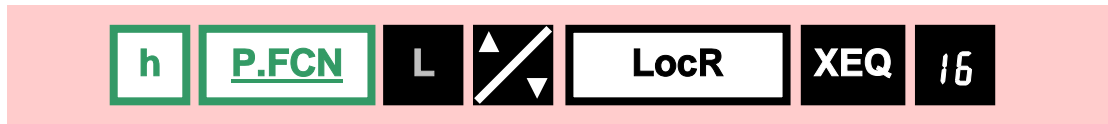


Just like variable – flags are all lost as soon as local data is cleared.

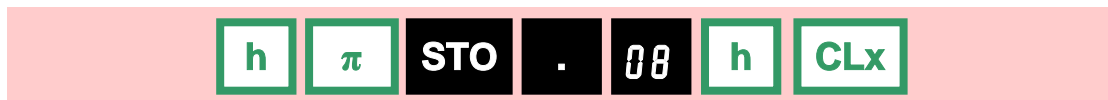
### Local Data - limitations

Local data can be used quite effectively but it does have some limitations – and all of these occur because local data is held in a dynamically allocated subroutine return stack. If the SBR stack changes, then the local data can and will be lost.

Consider the following example. Let us repeat the steps above, in the sense of first allocating 16 local data registers, and then storing Pi in register 8.



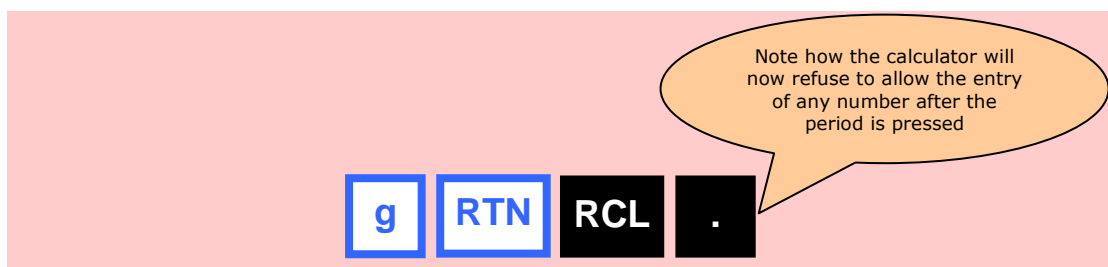
With the 16 local data register allocation made, let us now write the value of Pi into local data register 8 (remember we use the special dot notation to access these registers using .00 to .15). Then clear the X register



Next read back the local data register 8 to confirm that Pi has been written



Now execute the "RTN" function and try to read local data register 8 a 2<sup>nd</sup> time.



You will find that immediately following the execution of the return (RTN) command the calculator interface will refuse to allow the entry of anything following the period (.) character. As soon as the return (RTN) command is executed, the stack will have been reset and any and all local data cleared. At that point (for example) executing the "LocR?" query command in the "TEST" catalogue will confirm that zero registers remain allocated.

This example is a little contrived, because these local data functions really come into their own when executed within the body of a program and actually don't make a lot of sense when executed immediately via the keyboard.

#### Note

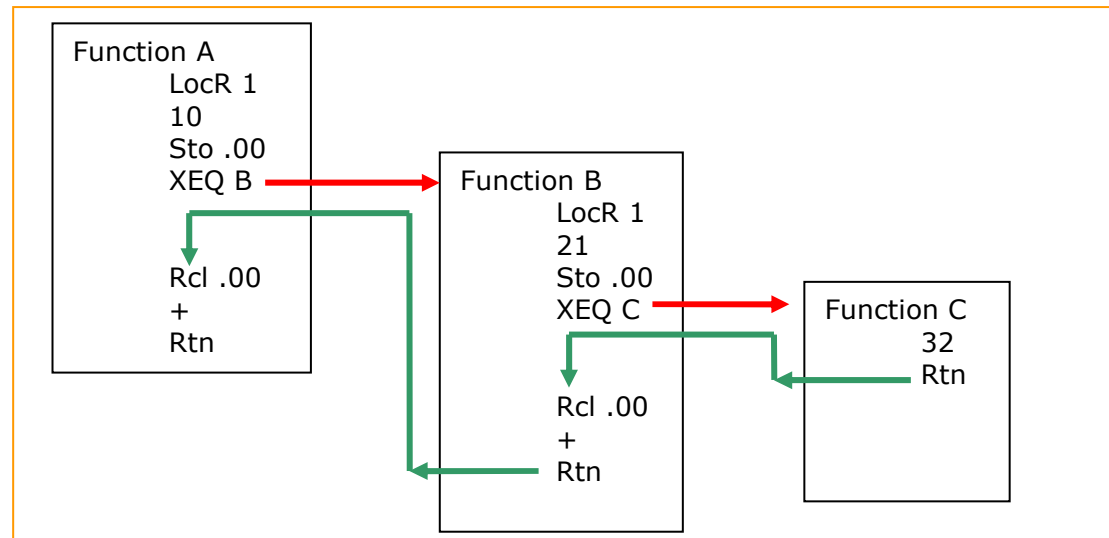


*Nonetheless, the example does highlight how these functions can and will fail if handled without care.*

When return (RTN) instructions are executed within the body of a program the code overseeing program execution will first clear the most recent allocation of local data, leaving any previously allocated local data ready for use. It will then return control to the calling function.



This means that local data can be allocated safely within nested function calls. Consider the following simply three function pseudo code designed to pass three values and to sum the three (ending up with the value  $10 + 21 + 32 = 63$ ).



*Figure 6 – Use of local data within nested functions*

Function (A) first allocates one local data register (which will be accessed using the dot notation .00). This function simply writes 10 into the local data register, and then executes (ie: calls) function (B).

Function (B) first allocates one local data register (which will be accessed using the dot notation .00) but which will not conflict with the same previously allocated local data register in function (A). Function (B) writes 21 into this local data register, and then executes (ie: calls) function (C).

Function (C) simply writes 32 into the X register, and returns control to function (B) which then reads the contents of its local register (previously loaded with 21) and adds this to the X register ( $21 + 32$ ) – leaving 53 in the X register. Function (B) then returns to function (A).

Function (A) reads the contents of its local register (previously loaded with 10) and adds this to the current X register ( $53 + 10$ ) leaving 63 in the X register. Function (A) then executes a RTN leaving the final result in the X register.

When function (A) executes its final return instruction, all remaining local data is cleared.

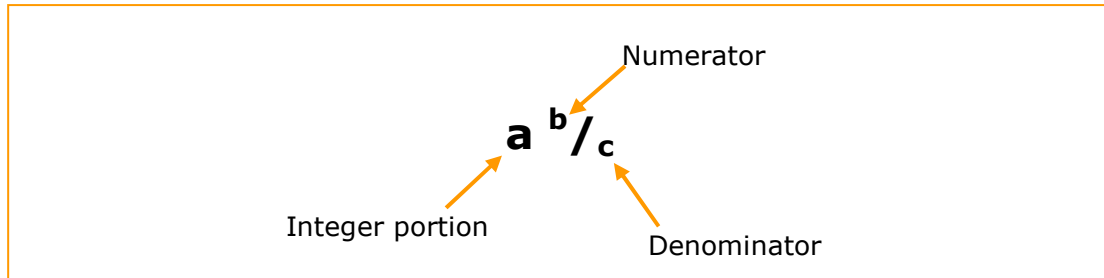
# Part 5

## WP 34S

## Fractions

## Fractions

A mixed fraction has the general form:-

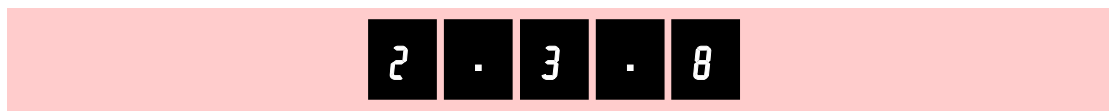


**Figure 7 – A Mixed fraction**

There are three types of fraction – a proper fraction has a numerator that is smaller than the denominator. An improper fraction is one where the reverse is true and a mixed fraction is made up from a whole number part plus a proper fraction (as shown in the diagram above).

Entering a fraction into the WP 34S involves typing the decimal point twice – first after the whole number (integer) part, and second between the numerator and denominator.

For example to enter  $2\frac{3}{8}$  use the following key strokes



To enter  $\frac{5}{8}$  use the following key strokes



To revert back to floating point mode, simply use the H.d command



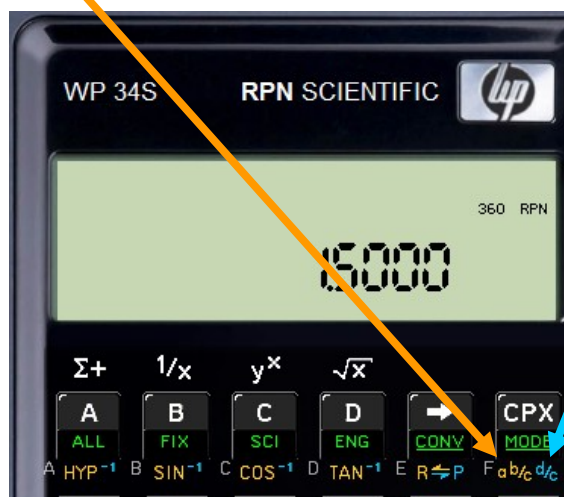
### Note



*On some HP calculators such as the HP 32Sii, you could enter a fraction such as  $\frac{5}{8}$  using the shortened key sequence 5..8. This key sequence won't work on the WP 34S.*

### ***Fraction modes – how the display works***

To switch from floating point mode into fractional mode, the calculator provides two keys at the top right of the keyboard. Both of these keys switch the calculator from a floating point mode into a fraction mode but the first key displays mixed and proper fractions ( $a^b/c$ ) and the second displays improper fractions ( $d/c$ ).

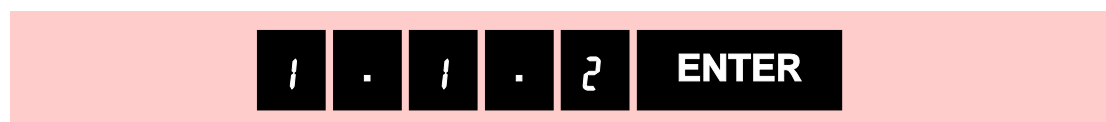


Improper fractions have a numerator larger than the denominator for example 48/23 and no whole number portion.

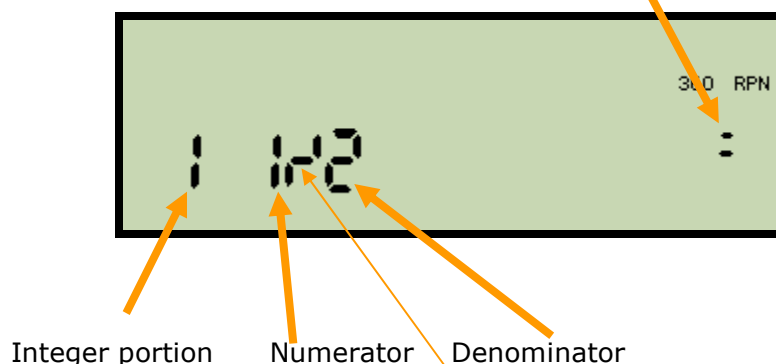
Converting a floating point number to a fraction is often an approximation. Sometimes the conversion will be exact but many times it will be slightly larger or smaller than the floating point value held in the calculator. The display announces this information using three guides. Consider the following three numbers...

### **Exact fraction conversions**

1.5 will convert precisely to  $1\frac{1}{2}$ . Enter this proper fraction using the following key sequence



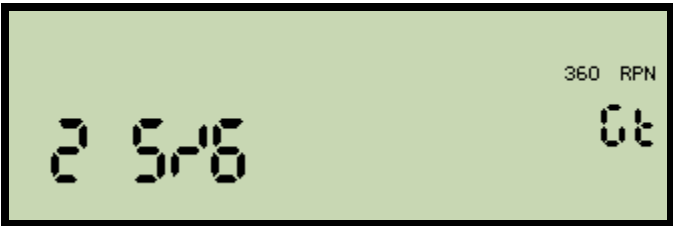
As this fraction can be converted exactly, the double line (equals) annunciator is displayed.



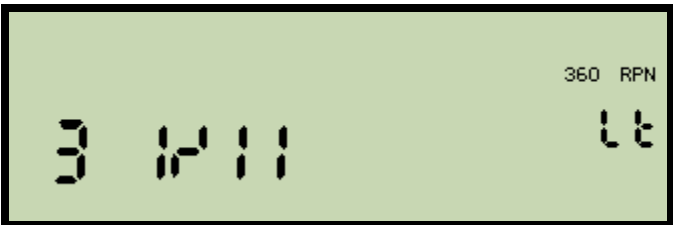
*Note the special symbol used to separate the numerator and denominator*

### Inexact fraction conversions

Set the calculator to work in proper fraction mode (**a<sup>b</sup>/c**) – and then consider the fraction  $\frac{34}{12}$ . Internally this improper fraction will be evaluated as 2.8333333333 and then converted to the proper fraction  $2\frac{5}{6}$ . The calculator will compare the converted fraction against the floating point value to the limit of its precision and will confirm if the fraction is greater than, or less than the floating point number. In this case, the fraction is (marginally) greater than the floating point number. Consequently the display will show the greater than annunciator "Gt":-



The same idea applies when dealing with a fraction such as  $\frac{34}{11}$ . Internally this improper fraction will be evaluated as 3.09090909091 and will convert to the proper fraction  $3\frac{1}{11}$ . The calculator will compare the converted fraction against the floating point value to the limit of its precision and will confirm that the fraction is slightly less than the floating point number. Consequently the display will show the less than annunciator "Lt":-



### Accuracy Indicators

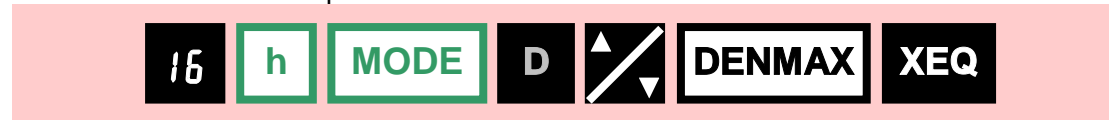
As shown above, the accuracy of the fractional display compared to the floating point value held by the calculator is confirmed by the following annunciators

The displayed fraction exactly matches the fractional part of the internal floating point number	=
The displayed fraction is slightly more than the fraction part of the internal floating point number. Note that the numerator difference will be no more than 0.5	Gt
The displayed fraction is slightly less than the fraction part of the internal floating point number. Note that the numerator difference will be no more than 0.5	Lt

In order to help explain this limit of accuracy in slightly more detail – assume for a moment that we fix the denominator of all fractions to be a particular value. We can do this using a pair of commands in the MODE catalogue called "DENMAX" and "DENFIX". When DENFIX is executed, it fixes the denominator to whatever is defined by DENMAX. The objective here is to artificially restrict the accuracy of the calculator so that we can then see the limits of accuracy.

Use the following key strokes to fix the denominator to 16. Note that the DENMAX command expects the new denominator to be in the X register before it is invoked.

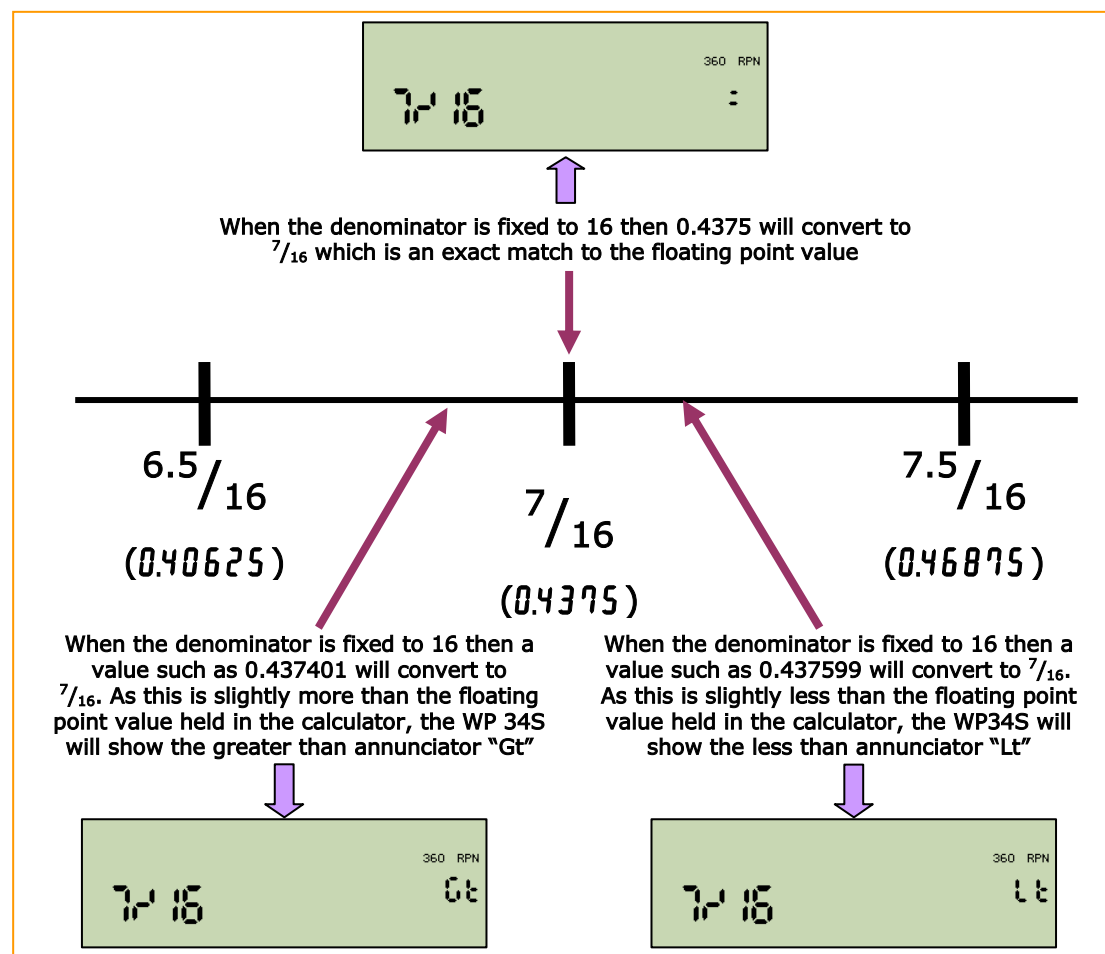
First set the maximum possible denominator to 16



Then execute the DENFIX command to make 16 the only denominator allowed.



Now, consider the band of values from  $6.5/16$  through to  $7.5/16$



#### Note

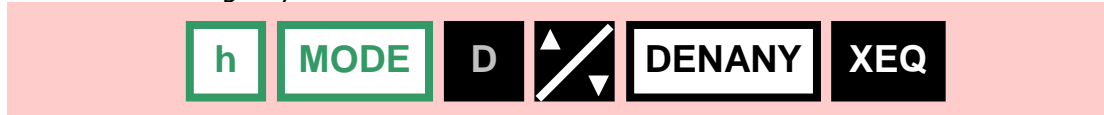


Following this example - don't forget to restore the default DENMAX value (using X=0) and also to switch the denominator fixed mode off by using the DENANY command. See the command sequence on the next page...

Use the following key strokes to restore DENMAX to its default value.



Use the following key strokes to switch off denominator fixed mode.



## Fraction Mode – Rounding

The WP 34S rounding function RND (secondary function on the 0 key) will round any fraction value which is slightly more or less than the floating point value. In effect rounding will make the fraction and the floating point values equal, and the equality symbol will replace either Gt or Lt.

## Fraction Mode – Limitations

The WP 34S fraction mode will handle numbers with absolute values that are greater than 0.0001 and less than 100000. The maximum denominator is 9999.

### Note



Therefore the fraction example quoted in the HP 32Sii user guide which converts  $e^{14}$  to a fraction won't work on the WP 34S because the absolute value of  $e^{14}$  exceeds 100000 ( $e^{14} = 1.1026 \times 10^6$ ).

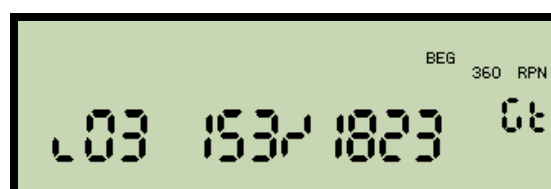
## Larger fractions

A displayed fraction may be too long to fit all digits into the display. By the time the numerator and denominator are displayed, the integer portion may end up being truncated. In order to cope with this, the calculator has a special annunciator used to signal that the integer portion has been truncated. You can then use the feature we looked at earlier to view all digits in the mantissa.

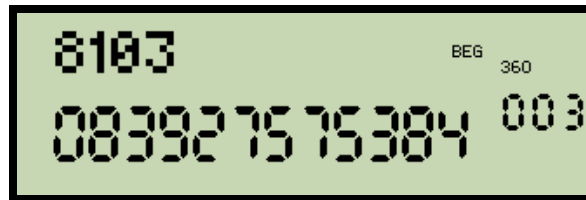
With the calculator using proper fractions ( $a^b/c$ ) consider finding e raised to the power 9. We would use the following key strokes



The resulting display estimates the fractional part of the result as 153/1823 and the integer part as 8103. As the combination of integer, numerator and denominator is too long for the display, the calculator truncates the integer portion to its two least significant digits and flags the fact with a small arrow head symbol at the bottom left



Use the special horizontal scroll key ◀( to view the truncated part. After pressing ◀( the entire mantissa and exponent will be visible



As explained earlier (Viewing All Mantissa Digits section), the mantissa display is a full display of 16 digits. The digits start at the top left and run for four places, and then continue on the left hand side of the next line down.

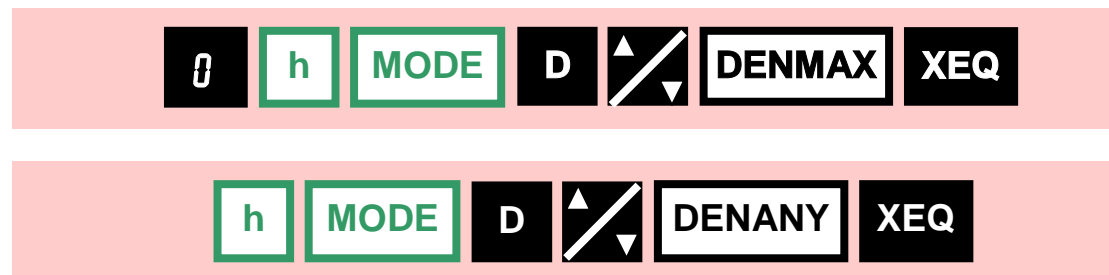
If the display showed an exponent of zero this would mean that the decimal point would sit between the first two digits on the top line. In this case, the exponent is actually 3, which means the decimal point will sit between the fourth digit (3) and the fifth digit (0).

As our fraction display showed 03 in its truncated integer portion then after viewing the entire mantissa we know that the full integer portion is 8103. Consequently the full fraction will be  $8103 \frac{153}{1823}$  (and we also know that this estimation will be slightly more than the floating point value held in the calculator because the display is showing the "Gt" annunciator).

### ***Controlling the fraction display format***

There are four important commands located in the MODE catalogue called DENMAX, DENANY, DENFAC and DENFIX which together allow the user to control the precise display of fractions. We look at each of these next.

If after using these modes you wish to return to the default fraction mode in the calculator, use the following two key sequences





## Setting the maximum denominator

For any fraction the denominator is selected based on a mode setting value in the calculator.

The calculator can work with any denominator up to a maximum value of 9999. The DENMAX command in the MODE catalogue can be used to control the maximum denominator that the calculator is allowed to use. For example to force the calculator to use denominators no greater than 16, the key strokes required would be...



Note that if you write 0 into the X register before calling DENMAX then the calculator will set the default denominator (9999). Alternatively, if you write 1 into the X register before calling DENMAX then the current DENMAX setting will be returned to the X register after the command executes.

## Setting the default fraction mode – max precision

The MODE catalogue function called DENANY sets the calculator to work with the default fraction format having maximum precision. Any denominator up to the value set by DENMAX can appear. For example if DENMAX=6 then setting DENANY allows denominators of 2,3,4,5 and 6. If DENMAX=0 then any denominator up to 9999 is permitted.

## Setting the fraction mode to employ factors

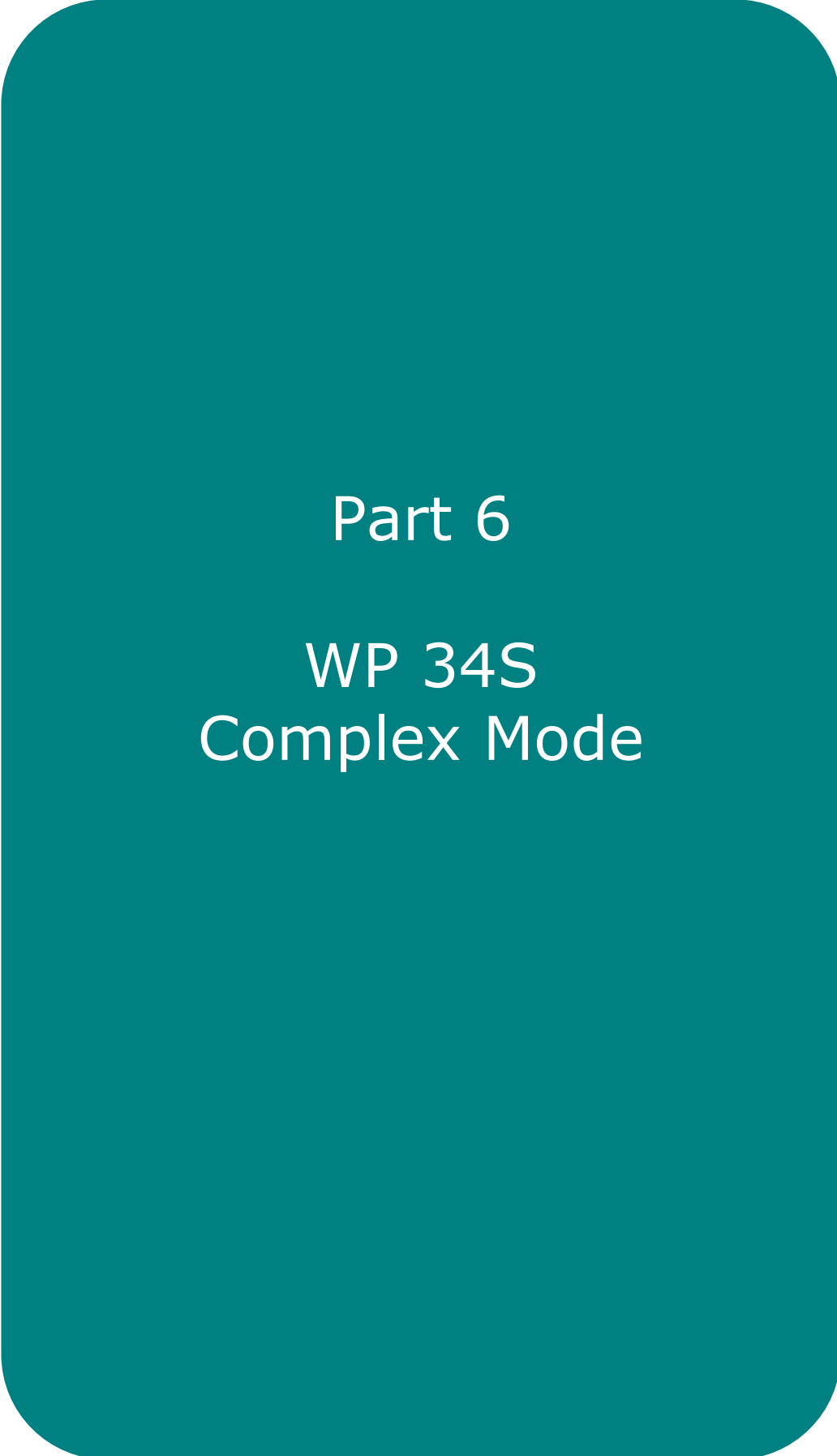
The MODE catalogue function called DENFAC sets the fraction mode to use only integer factors of the maximum denominator. For example if DENMAX=12 then setting DENFAC will allow denominators 2,3,4,6 and 12. Setting DENMAX=0 will permit any factor of 9999 to be used as the denominator.

## Setting a fixed fraction denominator

The MODE catalogue function called DENFIX sets the fixed fraction mode where the one and only denominator allowed is the value defined by DENMAX. If DENMAX is set to its special default value of 0, then the single fixed denominator permitted will be 9999

## Using fractions in programs

You can type fractions into programs in the normal way (a.b.c or ..b.c) and when entered in that format, program execution will force the display mode to change to fraction mode.

A large teal rounded rectangle with a thin red border, serving as a background for the section header.

# Part 6

## WP 34S

### Complex Mode

## Complex modes of operation - introduction

The WP 34S supports many complex operations. The CPX key (top right) is used as a prefix for calling complex functions for example finding the complex reciprocal requires the following key strokes (assume the local label B is currently assigned as a program label)



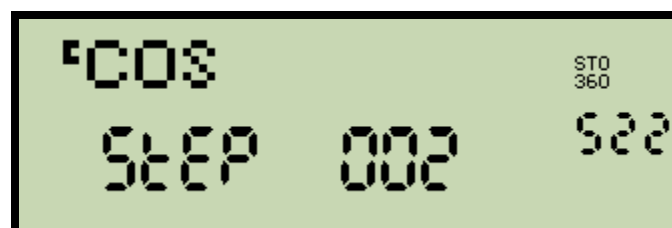
As soon as you press CPX, the calculator prepares for a complex operation – and displays a small c annunciator as follows:-



Once the operation is completed (in this case **f**, **reciprocal**) the display confirms that a complex result is present by a larger C in the top left.



Entering such a command in program mode will show the command with a top left "c" annunciator in the program listing – as in this example where Step 2 of the program has had a complex cosine instruction inserted. The resulting display would appear as:-



The WP 34S has operations for complex arithmetic (add, subtract, multiply, divide), complex trigonometry (Sin, Cos, Tan,  $\text{Sin}^{-1}$ ,  $\text{Cos}^{-1}$ ,  $\text{Tan}^{-1}$ ) and the mathematic functions  $1/x$ , ABS, FP (fractional part), IP (integer part), RND,  $x!$ ,  $x^2$ ,  $x^3$ ,  $\sqrt{x}$ ,  $\sqrt[3]{x}$ ,  $+/-$ , log and exponential functions with bases 10, 2 and e as well as hyperbolic trig functions (and the inverses).

## Complex mode - extra function catalogue

The WP 34S provides access to a range of special functions that work in the complex domain. You access these, by opening a special catalogue – and keeping in mind the function of the CPX key (to act as a prefix for any complex commands) you use the X.FCN button but prefixed by CPX to open this particular catalogue of complex functions.

CPX

h

X.FCN

## Complex number format

Complex functions in the WP 34S use Cartesian coordinates throughout. The standard format is:-

$$X + iy.$$

Each complex number occupies two adjacent registers – the upper holding the imaginary part, and the lower holding the real part. Therefore - when entering complex values onto the stack you enter the imaginary part first (which will end up in Y), followed by the real part (which will be in X).

### Note



*You can always convert to a polar view of a complex number, but remember to always convert back to Cartesian form before performing any complex functions.*

For example – the complex number  $(1 + i2)$  can be converted to polar coordinates via the following key sequence

2

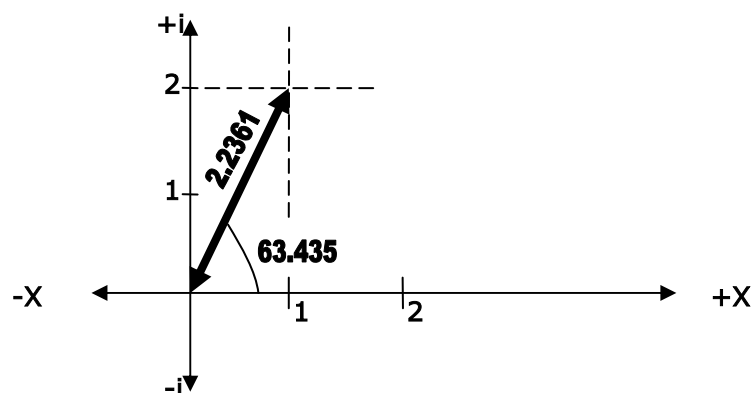
ENTER

/

g

R↔P

The polar coordinates are returned with the magnitude  $r$  in the X register, and the angle  $\theta$  in the Y register (in this case  $r=2.2361$ , and  $\theta = 63.435^\circ$ ).



Use the reverse ( $\rightarrow$ rectangular) function to revert to Cartesian coordinates.



### ***Entering complex numbers and invoking complex functions***

When entering a complex number, first type the imaginary part then press ENTER and then type the real part giving



$$x + iy.$$

Complex numbers are handled by entering the imaginary and real parts as separate numbers. To enter two complex numbers, you would enter four separate numbers. To perform a complex operation, you prefix the operation with the key CPX.

For example to add the two complex numbers  $(3+i5)$  and  $(2+i4)$  you would use the following key strokes



After this key sequence, the result will be stored in the X register (holding the real part) and the Y register (holding the imaginary part). The display initially shows the real part, but flags that this is a complex result with a C in the top left. The user would X exchange Y to read the imaginary part (or use roll ie: R↓)

The real part of the result is shown first and the calculator confirms it is a complex result via the C at the top left.	Pressing the X exchange Y key then reveals the imaginary part of the sum.
	

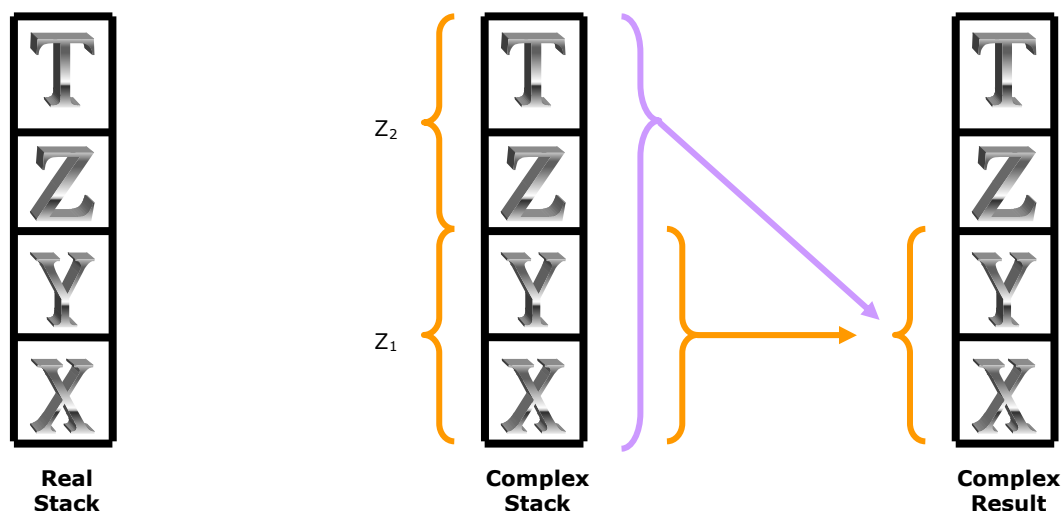
$$i.e. (3+i5) + (2+i4) = (5 + i9)$$

### The complex stack

The complex stack is simply the original X,Y,Z,T stack split into two double registers capable of holding two complex numbers i.e.

$$(Z_{1X} + i Z_{1Y}) \text{ and } (Z_{2X} + i Z_{2Y})$$

Looking at the stack itself – the format of our standard RPN real mode stack is shown on the left – consisting of the four registers X,Y,Z and T.



When dealing with complex numbers we can either store  $Z_1$  in the first two stack registers X and Y – in which case functions operating on just one complex number for example Sin, Cos, Tan and  $1/x$  – which can be generally written as  ${}^c f(Z_1)$ , will leave the single complex result in the same two registers X and Y.

Alternatively we can store the combination of two complex numbers  $Z_1$  and  $Z_2$  in the stack using registers X and Y to hold  $Z_1$ , and registers Z and T to hold  $Z_2$ . In which case functions operating on two complex numbers at a time for example add, subtract, multiply, divide and  $y^x$  – which can be generally written as  ${}^c f(Z_1, Z_2)$  will leave the single complex result in the X and the Y register.

Putting this more formally the WP 34S works generally as follows. If some arbitrary real function  $f$  operates on...

- ...one real number  $x$  only, then its complex brother  ${}^c f$  will operate on the complex number  $x + iy$  (ie: the real part will be in stack register  $x$  and the imaginary in stack register  $y$ )
- ...a single register eg:  $R_{nn}$ , then its complex brother  ${}^c f$  will operate on the pair of registers  $R_{nn}$  and  $R_{nn+1}$
- ... $x$  and  $y$ , then  ${}^c f$  will operate on  $x, y, z$  and  $t$

In the case where a real function using a single argument replaces stack register  $x$  by the result of  $f(x)$  then by contrast the same one argument *complex* operation will replace stack register  $x$  by the real part of the function, and will replace stack register  $y$  with the imaginary part of the complex function  ${}^c f(x_c)$

Similarly a real function which uses two arguments will generally replace stack register x with the result of  $f(x,y)$ . By contrast a two argument complex operation will replace stack register x by the real part of the function  ${}^cf(x_c,y_c)$ , and will replace stack register y with the imaginary part of the complex function  ${}^cf(x_c,y_c)$

**Note**



*Using complex number operations on the WP 34S doubles the memory requirement compared to real operations. However – it is worth keeping in mind that the stack size of the WP 34S can be optionally extended from four to eight registers (see the function **SSIZE8**, and its complement **SSIZE4** in the **MODE** catalogue). Setting a stack size of 8 means the calculator has a stack capable of holding four full complex numbers.*

## Complex Operations

Use complex operations just as you would real operations, but precede the operator with the CPX key



### Operation on one complex number

To perform an operation with one complex number

1. Enter the complex number  $Z$ , composed of  $(x+iy)$ , by keying in  $y$  ENTER  $x$
2. Select the complex function

To compute	Use keystrokes
Change sign, $-z$	<b>CPX</b> <b>+/-</b>
Inverse $1/z$	<b>CPX</b> <b><math>1/x</math></b> <sup>*1</sup> Or... <b>CPX</b> <b>f</b> <b><math>1/x</math></b>
Natural Log, $\ln z$	<b>CPX</b> <b>g</b> <b>LN</b>
Natural antilog, $e^z$	<b>CPX</b> <b>f</b> <b><math>e^x</math></b>
Sin $z$	<b>CPX</b> <b>f</b> <b>SIN</b>
Cos $z$	<b>CPX</b> <b>f</b> <b>COS</b>
Tan $z$	<b>CPX</b> <b>f</b> <b>TAN</b>

<sup>\*1</sup> Use the alternative key strokes if the "B" key is used as a label in the current program

**Operation on two complex numbers**

To perform an operation with two complex numbers

- 1. Enter the first complex number  $Z_1$ , composed of  $(x_1+iy_1)$ , by keying in  $y_1$   
ENTER  $x_1$
- 2. Enter the second complex number  $Z_2$ , composed of  $(x_2+iy_2)$ , by keying in  $y_2$  ENTER  $x_2$
- 3. Select the arithmetic function

To compute	Use keystrokes		
Add $Z_1 + Z_2$	CPX	+	
Subtract $Z_1 - Z_2$	CPX	-	
Divide $Z_1 \div Z_2$	CPX	$\div$	
Multiply $Z_1 \times Z_2$	CPX	$\times$	
Power $y^x$	CPX	$y^x$	<sup>*1</sup> Or... CPX f $y^x$

<sup>\*1</sup> Use this alternative key strokes if the "C" key is used as a label in the current program



# Part 7

## WP 34S

### Unit Conversions

























## Unit conversions - introduction

HP calculators of old have always included a useful range of unit conversions. These are functions which can be used to convert one measurement type or unit to another linked type.

For example pressure can be described in bar, or in pounds per square inch. Similarly numbers could be binary or hexadecimal, length could be metric or imperial and temperature could be in Fahrenheit or Centigrade. It is useful to be able to convert one to the other

### Immediate Conversions

The WP 34S provides a small subset of quick handy conversions using the right arrow key (top right of the keyboard). The following eight conversions are possible using this "quick" mechanism.

To convert	Use keystrokes
Assuming that the X register contains a value of decimal hours or degrees this will converted X into the format hhhh° mm' ss.dd	  
Assuming the X register contains a value in the format hhhh° mm' ss.dd, this function converts to decimal hours / degrees.	  
Based on whatever angular mode is active (radians, gradians or degrees) this function converts the contents of the X register to degrees.	  
Based on whatever angular mode is active (radians, gradians or degrees) this function converts the contents of the X register to radians.	  
Based on whatever angular mode is active (radians, gradians or degrees) this function converts the contents of the X register to gradians.	  
<b>Temporarily</b> converts the integer portion of the X register into binary base 2 until the next keystroke (when the X register contents will revert to their value before the conversion)	  
<b>Temporarily</b> converts the integer portion of the X register into octal base 8 until the next keystroke (when the X register contents will revert to their value before the conversion)	  
<b>Temporarily</b> converts the integer portion of the X register into hexadecimal base 16 until the next keystroke (when the X register contents will revert to their value before the conversion)	  

#### Note



Observe that three of these conversions are temporary (to-binary, to-octal and to-hexadecimal). After the conversion is displayed in the X register, pressing any key will cause the conversion display to clear.

## Functional based conversions

The WP 34S provides a substantial number of conversions within the CONV catalogue (top right of the keyboard). Navigation works in a similar way to all the other catalogues we've used thus far with two interesting differences. First the conversion process is immediate. Second there is a handy key stroke which inverts (ie: undoes) any conversion and which closes the catalogue

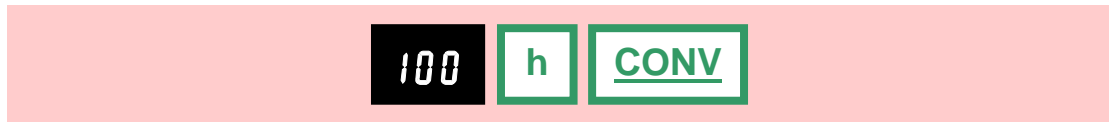
### Note



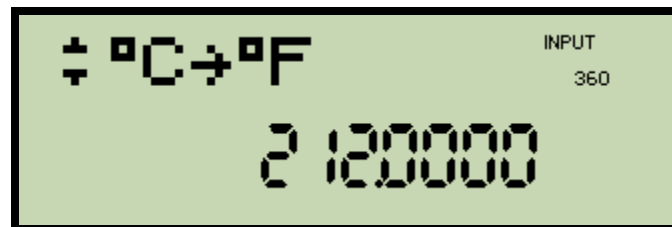
*For this example – keep in mind...*

1. 100° centigrade is approximately 212° Fahrenheit
2. 100° Fahrenheit is approximately 37.7778° centigrade

Enter 100 into the X register and invoke the CONV catalogue using the following key strokes

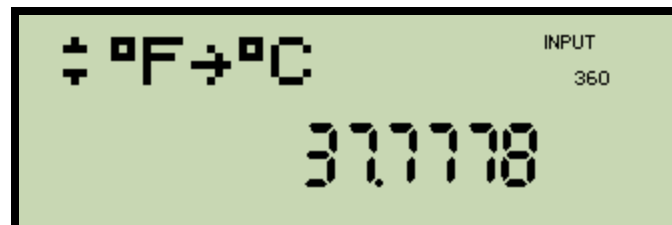


The display will then show the immediate conversion of the value in the X register using the currently selected conversion function – which happens to be centigrade to Fahrenheit ie:-



If you click the backspace key or the exit key, then the conversion (in this case °C to °F) will be cancelled and the CONV catalogue closed (at which point the X register will once again contain 100.0000).

Note also that if you scroll down through the list of conversions using the scroll arrow (▲▼) keys, each new conversion runs immediately. So for example hitting the down arrow will take you to the following alternate conversion (which is actually the inverse)



We started with 100 degrees in X, each conversion took the original X content and converted it to °F in the first conversion, and to °C in the second.

## Accepting a Conversion

To accept the conversion so that the source value in the X register is changed simply press ENTER or XEQ.

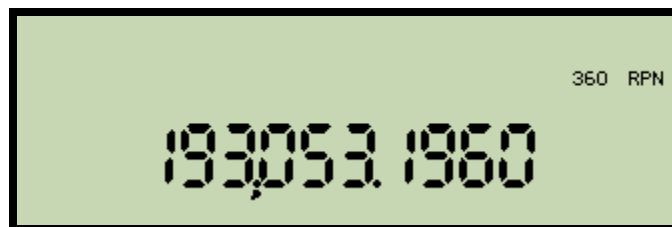
## Combining Conversions

The tyre pressure of my vehicle is 28psi – what is the equivalent pressure in bar

First convert the tyre pressure from PSI into pressure absolute (Pa). Note that we can easily find this conversion by typing the first two letters (PS) of the conversion, and press enter to accept



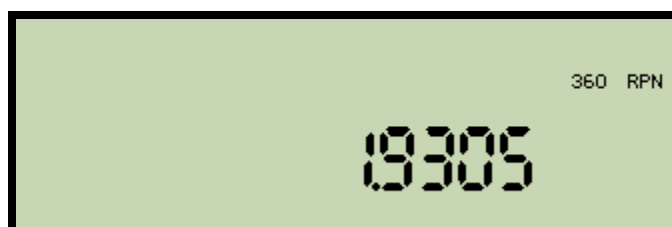
The resulting display will show the 28 PSI converted to Pa



Next convert the PA value to BAR using the Pa→bar conversion. With the catalogue open, press the “P” letter to select all conversions starting with P, and then press the scroll down key once to get to the PA to BAR conversion. Press ENTER to accept.



The resulting display will show the 193053.1960Pa converted to bar

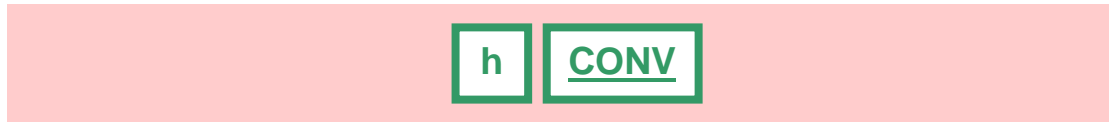


So a tyre pressure of 28PSI is roughly equivalent to 1.9BAR

### Quick conversion inversion

With the calculator setup as above, if you wished to invert the above BAR pressure and get back to the Pa pressure, there is a special key stroke trick you can use. First select the right conversion in the open catalogue, and then press the key sequence **f** B. This will perform the inverse of the selected conversion.

In this case - first open the CONV catalogue, and note that it automatically opens showing the previously invoked conversion Pa→bar



To invert this conversion (and close the catalogue and leave the previous result (Pa) in X), then simply use the following key strokes

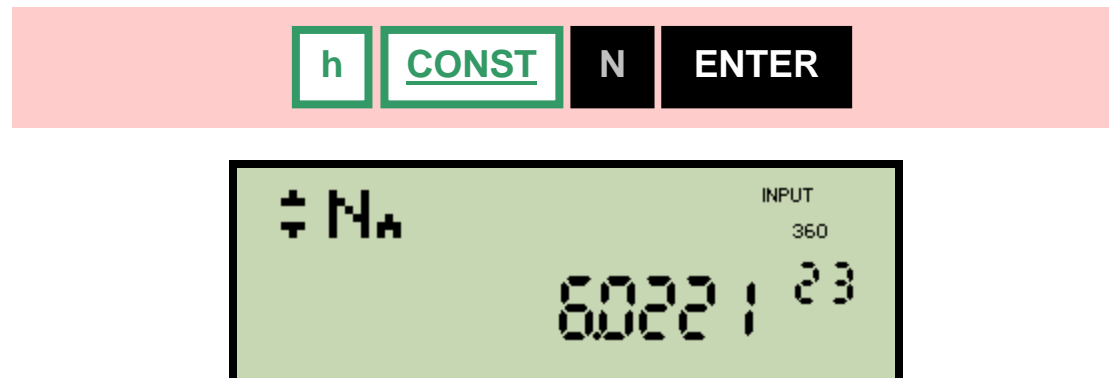


# Part 8

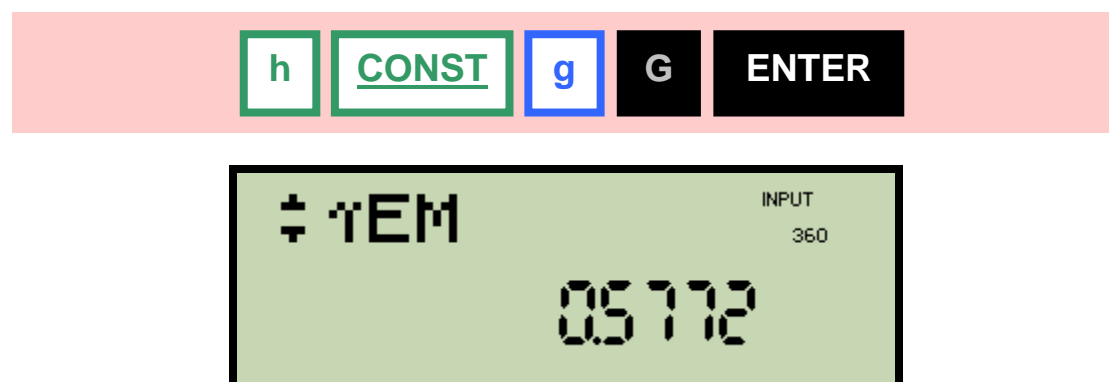
## WP 34S Constants

## Constants – Introduction

The WP 34S has a very rich set of constants. The complete set of constants is located in the CONST catalogue and accessed in the normal way. For example to find the value of Avogadro constant  $N_A$  and to place it in the X register – use the following keystrokes.



To find the Euler-Mascheroni Constant which uses the Greek letter gamma, use blue shift G to highlight all constants starting with the Greek letter equivalent of "G". Press ENTER to insert the constant into the X register



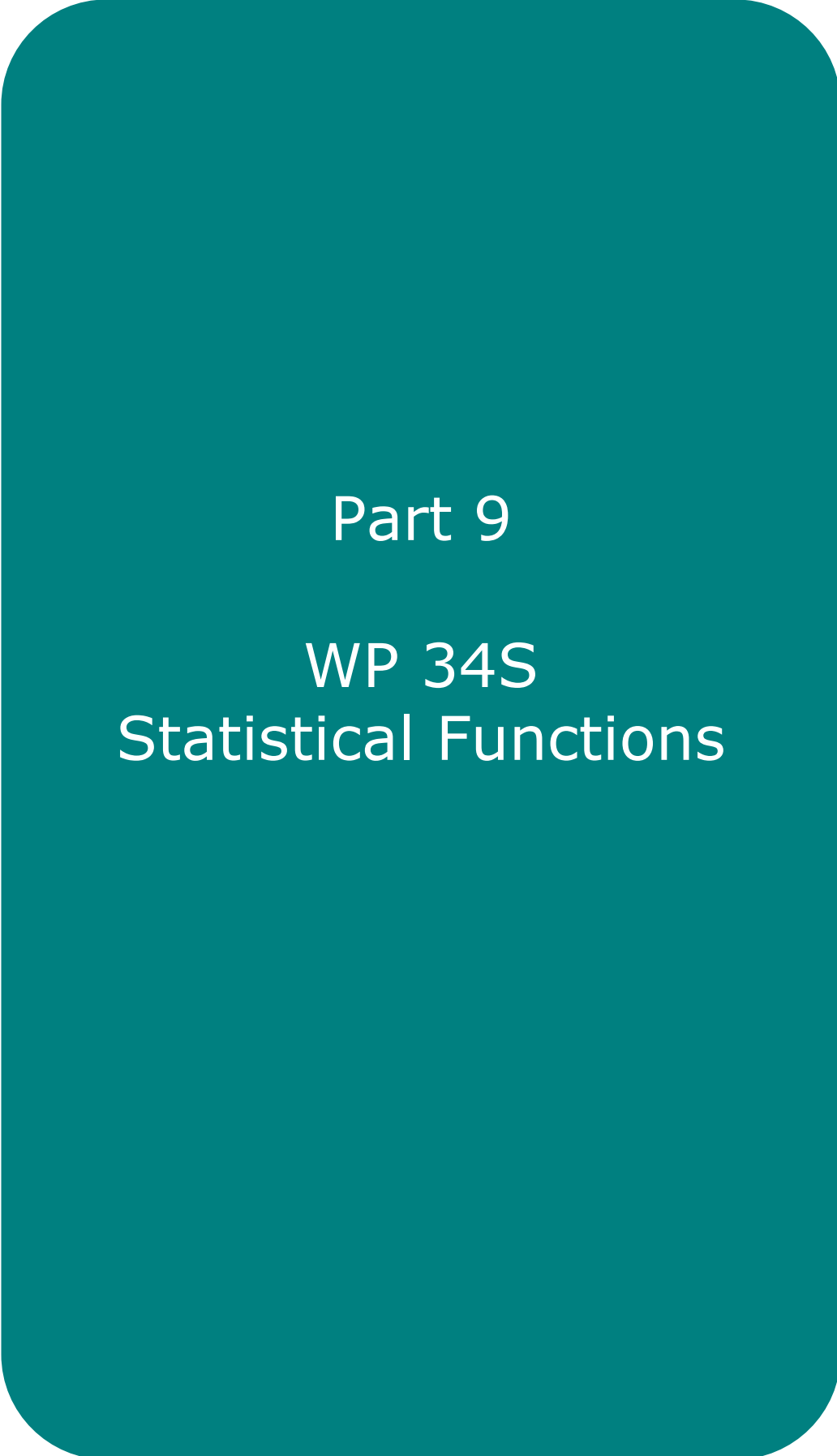
Note that within this catalogue, Eulers number constant  $e$  (2.718) should not be confused with the charge of an electron ( $e$ ). Eulers is assigned the name  $eE$  in the CONST catalogue.

### *Use of constants in the complex domain*

Note that you can prefix the constant catalogue with the CPX key in order to enter the constant into X, and a zero into Y.



Would place 2.7183 into the X register, and zero into the Y register

A large teal rounded rectangle with a thin red border, serving as a background for the section header.

# Part 9

## WP 34S

### Statistical Functions



## Statistical Functions - Accumulations

Pressing the  $\Sigma+$  key computes certain important sums and products of the input values provided in the X and Y registers. Before you start to calculate accumulations with a new set of X, Y values you first need to clear the accumulation registers using the following keystrokes



Then do the following for each pair of X and Y values in your data set

1. Key in the Y value
2. Press ENTER
3. Key in the X value
4. Press  $\Sigma+$

### Note



*Remember, if local label A is not used in the current program then the A key will be assigned the  $\Sigma+$  function automatically. If the key is being used by a local program label, then you can still access the  $\Sigma+$  function using  $h \Sigma+$  (a secondary function of the + (add) key)*

If the problem you're working on only involves one variable (x) instead of two (x,y) then the procedure for entering values is similar with one important point to note. Before you start, clear the statistical storage and clear the Y register to zero. Then start entering all the values in the list. **You must clear the Y register because a non zero value in Y will cause a domain error for some one-number statistical calculations.**

Each time you press  $\Sigma+$  a new data item is added to the list of values in the calculator and the index of that item is shown in X starting from index 1.

## Deleting and Correcting Data

If you key in the incorrect value, and have not yet pressed  $\Sigma+$  then simply clear X and rekey. If you've already pressed  $\Sigma+$ , then press  $\Sigma-$  (note that the index in the X register will reduce by 1), rekey your values and press  $\Sigma+$ .

## Finding the Mean

Pressing the f shifted  $\bar{x}$  key...








...computes the arithmetic mean (average) of X and Y values accumulated in the statistical registers. When you use this function, the following occurs.

1. The contents of the stack registers are lifted.
2. The mean of the X values is calculated using the data accumulated in the sigma registers according to the formula  $\bar{x} = \Sigma x / n$ . The resulting value appears in the displayed X register
3. The mean of the Y values is calculated using the data accumulated in the sigma registers according to the formula  $\bar{y} = \Sigma y / n$ . The resulting value is available in the Y register of the stack.

Consider the following example. The table below shows a chart of daily high and low temperatures for a winter week somewhere in the world. What are the average high and low temperatures for the week selected?

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
High	6	11	14	12	5	-2	-9
Low	-22	-17	-15	-9	-24	-29	-35

Use the following keystrokes to calculate the averages

Keystrokes	Display
 	Display will show what was in the X reg
6 ENTER 22 +/- Σ+	1.0000
11 ENTER 17 +/- Σ+	2.0000
14 ENTER 15 +/- Σ+	3.0000
12 ENTER 9 +/- Σ+	4.0000
5 ENTER 24 +/- Σ+	5.0000
2 +/- ENTER 29 +/- Σ+	6.0000
9 +/- ENTER 35 +/- Σ+	7.0000
 	-21.5714 (the average low temp)
	5.2857 (the average high temp)

The average of the low temperatures is -21.5714, and the average for the higher temperatures is 5.2857.

## Standard Deviation

Pressing the **S** key (secondary function of digit 5) computes the standard deviation (a measure of dispersion around the mean) of the accumulated data. The formulas used by the WP 34S to compute  $S_x$ , the standard deviation of the accumulated x values, and  $S_y$ , the standard deviation of the accumulated y values are...

$$S_x = \sqrt{\frac{n \Sigma x^2 - (\Sigma x)^2}{n(n-1)}}$$

$$S_y = \sqrt{\frac{n \Sigma y^2 - (\Sigma y)^2}{n(n-1)}}$$

These formulas give the best estimate of the population standard deviation from the sample data. Consequently the standard deviation given by these formulas is termed by convention the *sample* standard deviation

When you press **g s** to compute the standard deviation the following happens

1. The stack lifts
2. The standard deviation of the X values ( $S_X$ ) is calculated using the data accumulated in the sigma registers using the formula shown above. The result is shown in the X register.
3. The standard deviation of the Y values ( $S_Y$ ) is calculated using the data accumulated in the sigma registers using the formula shown above. The result is held in the Y register.

Example: A new test is developed that measures the bearing reliability of shopping trolley wheels. To evaluate its effectiveness the test is administered to 1000 trolleys in a store in London and then 10 of the 1000 tests are taken, and an estimate made of the standard deviation of all the results from the sample of 10.

The scores for the 10 samples are: 52, 38, 97, 81, 64, 78, 99, 26, 33 and 12.

What is the standard deviation?

Keystrokes	Display
<b>0 ENTER</b>	0.0000 (clear the Y register)
<b>g</b> <b>CLΣ</b>	0.0000
52 Σ+	1.0000
38 Σ+	2.0000
97 Σ+	3.0000
81 Σ+	4.0000
64 Σ+	5.0000
78 Σ+	6.0000
99 Σ+	7.0000
26 Σ+	8.0000
33 Σ+	9.0000
12 Σ+	10.0000
<b>g</b> <b>S</b>	30.4923

Based on a sample of 10, the standard deviation estimate for the bearings failure rate on 1000 shopping trolleys is 30.4923.

#### Note



*Did you see "Domain Error" when you tried to compute the standard deviation? Refer to the notes above for one number functions – and note that a nonzero value in the Y register (before you started to enter the data) is the problem. Make sure you clear the Y register before starting the calculation.*

When the data constitutes not just a sample of a population but rather *all* of the population, the standard deviation of the data is the *true* population standard

deviation (denoted  $\sigma$ ). The formula for the true population standard deviation differs by a factor of  $[(n-1)/n]^{1/2}$  from the formula used by the WP 34 **S** function. The difference between the values is small, and for most applications can be ignored. Nevertheless, if you want to calculate the exact value of the population standard deviation for an entire population you can easily do so with just a few keystrokes. Simply add (using the  $\Sigma+$  key) the mean ( $\bar{x}$ ) of the data list to the data list itself, and then compute the standard deviation using **S**. The result will be the true population standard deviation of the original data.

In the previous example we would use

<b>f</b> <b><math>\bar{x}</math></b>	58.0000
<b><math>\Sigma+</math></b>	11.0000
<b>g</b> <b>S</b>	28.9275

The standard deviation for all results in the test is: 28.9275

## Linear Regression

Linear regression is a statistical method for finding a straight line that best fits a set of data points thus providing a relationship between two variables. After a group of points have been entered you can calculate the coefficients of the linear equation  $y=mx+c$  using the least squares method by using the L.R. function built into the STATS catalogue. It goes without saying that there must be a minimum of two data points stored in the calculator before a least squares line can be fitted.

When you use linear regression the following occurs.

1. The contents of the stack registers are lifted
2. The slope M of the least squares line of the data is calculated using the following equation and left in the Y register of the stack

$$M = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$$

3. The Y axis intercept C of the least squares line of the data is calculated using the following equation and left in the X register of the stack

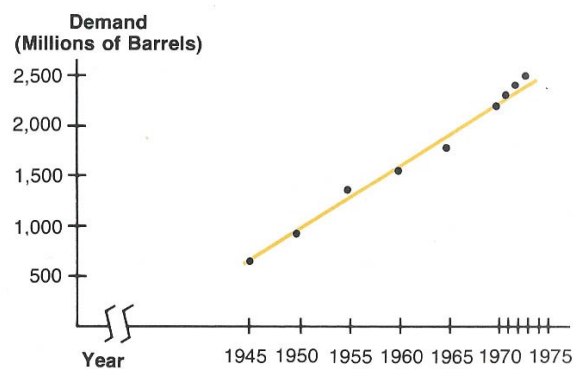
$$C = \frac{\sum y \sum x^2 - \sum x \sum xy}{n \sum x^2 - (\sum x)^2}$$

To obtain the slope, simple press the x exchange y key or use roll.

For example – the following table shows the consumption of barrels of oil in the UK over the years 1945 to 1973. Find the slope and Y intercept of a least squares solution for the consumption against time.

Year	1945	1950	1955	1960	1965	1970	1971	1972	1973
Barrels (x10 <sup>6</sup> )	696	994	1330	1512	1750	2162	2243	2382	2484

You could start by graphing the data as follows. Based on this you could estimate the  $y=mx+c$  relationship which forms the best fit.



Compute the least squares solution as follows:-

Keystrokes	Display
<b>g</b> <b>CLΣ</b>	0.0000
696 ENTER 1945 Σ+	1.0000
994 ENTER 1950 Σ+	2.0000
1330 ENTER 1955 Σ+	3.0000
1512 ENTER 1960 Σ+	4.0000
1750 ENTER 1965 Σ+	5.0000
2162 ENTER 1970 Σ+	6.0000
2243 ENTER 1971 Σ+	7.0000
2382 ENTER 1972 Σ+	8.0000
2484 ENTER 1973 Σ+	9.0000
<b>h</b> <b>STAT</b> <b>L.R.</b> <b>XEQ</b>	- 118290.6295 (Y intercept)
<b>X↔Y</b>	61.1612 (slope)

So the relationship during these years has a best fit to the equation

$$Y = 61.1612(X) - 118290.6295$$

#### Note



As a check – consider the year (X) value of 1955. The equation predicts a consumption value of 1279.5 barrels which is very close to the actual value of 1330.

## Linear Estimation

With data entered into the accumulation registers a predicted value for Y (denoted as a  $\hat{y}$ ) can be calculated. Similarly a predicted value for X (again denoted as  $\hat{x}$ ) can be calculated.

Whereas the Y estimation is available on the keyboard (secondary function on the 6 key), the less commonly required X estimation is available within the STAT catalogue...



For example – with the data intact from the previous problem, if you wished to predict the demand for oil in the years 2012 and 2014, you would first key in the value of 2012 as the known X value, and then press  $\hat{y}$

Note the display shows the prediction for the year 2012 at slightly less than 5 billion barrels.

Keystrokes	Display
2012 $\hat{y}$	Linear 360 4765.7860

Similarly – if we wanted to know when the demand for oil would exceed 3500 million barrels, we would key in 3500 (the new value for y) and select the  $\hat{x}$  estimation in the stats catalogue ie:

Keystrokes	Display
3500	
$\hat{h}$ STAT X $\hat{x}$ XEQ	Linear 360 1991.3041

In other words, demand for 3500 barrels will occur sometime between 1991 and 1992.


## Correlation Coefficient

Both linear regression, and linear estimation presume that the relationship between X and Y data values can be approximated to some degree by a straight line (linear) function.

You can use the  $r$  correlation coefficient function (a secondary function on the 6 key) to assess how well or badly your data fits a straight line. The coefficient ranges from +1 to -1, where  $r=+1$  means that the data falls exactly onto a straight line with positive slope. With  $r=-1$  the data falls exactly onto a straight

line with negative slope. When  $r=0$  the data cannot be approximated at all by a straight line.

To calculate the correlation coefficient for the oil barrel example – use the following key strokes

Keystrokes	Display
	Linear 09931

This value confirms that the data correlates very closely to a straight line with positive slope (as we know it does from the graph).

## Normal Distribution

There are two useful functions available on the keyboard for problems dealing with normal distributions. Ie:  $\Phi$  and  $\Phi^{-1}$ . The function ( $\Phi$ ) computes the area under the standard normal distribution curve to the left of X. Its inverse ( $\Phi^{-1}$ ) computes X given the area under the standard normal distribution curve to the left of X using an iterative algorithm.

### Note



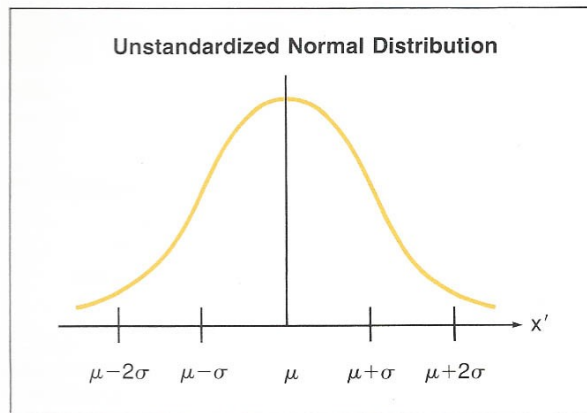
*It is quite revealing to compare the execution time of the inverse algorithm for  $\Phi^{-1}$  running on a 1978 HP32E (LED) calculator against that of the WP 34S. After a barely perceptible delay, the result is available on the WP 34S display. By contrast the HP32E takes a full 15 seconds to compute the result (but it does have the upside of showing a vivid display on the LED's as it runs).*

The area under the standard normal distribution curve to the left of X is a measure of the probability or the frequency of the occurrence for all values less than or equal to X

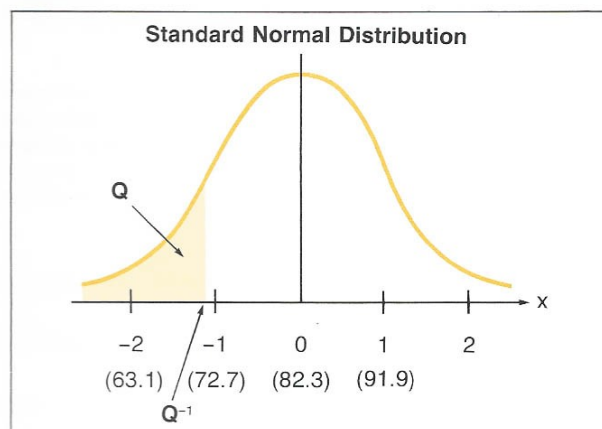
The number that is input to the function ( $\Phi$ ) or returned by the inverse function ( $\Phi^{-1}$ ) is the standard variable for the standard normal distribution. Given the mean value  $\mu$  and the standard deviation value  $\sigma$  of a normal distribution the following formulas give the relation between X (the standard variable) and X' the unstandardised variable of the problem.

$$X = \frac{X' - \mu}{\sigma} \qquad X' = \sigma X + \mu$$

The standard normal distribution has  $\mu=0$  and  $\sigma=1$  as shown in the following diagram...



Consider this example. The distribution for final exam scores for 200 apprentices has a mean  $\mu=82.3$ , and a standard deviation  $\sigma=9.6$ . What proportion of the apprentices receive passing scores of 70 or higher



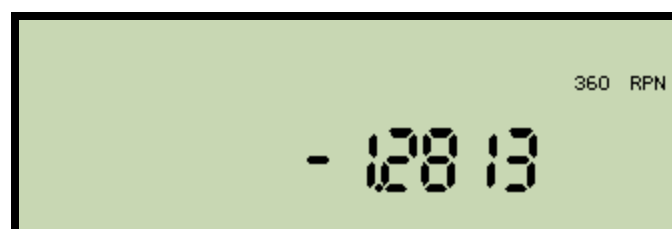
First find  $X$  using the formula:

$$X = \frac{X' - \mu}{\sigma} = \frac{70 - 82.3}{9.6}$$

Use the key strokes





The display will show








This value for X indicates that the score of interest (70) lies 1.2813 standard deviation to the left of the mean score 82.3. Now proceed with the rest of the problem.








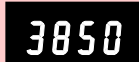



Keystrokes	Display
 	0.1001 <small>360 RPN</small>

This display confirms the proportion receiving scores *less than* 70. We can invert the result using the following keystrokes.

Keystrokes	Display
  	0.8999 <small>360 RPN</small>

Close to 90% of the apprentices passed the test.

Consider this alternate example. The monthly income in households in Farnham can be described by a normal distribution with a mean of £12400 and a standard deviation of £3850. What is the income level exceeded by the highest 10% of the households?

Keystrokes	Display
First enter the highest tenth of households	
 	0.1000 <small>360 RPN</small>
Adjust to make it the proportion less than the income level required. This is necessary because the $\Phi^{-1}$ function assumes that the number in the X register represents the area to the left of the standard variable.	
  	0.9000 <small>360 RPN</small>
Then calculate the number of standard deviations above the mean income	
 	1.2816 <small>360 RPN</small>
Multiply by the standard deviation	
 	4933.9735 <small>360 RPN</small>
And finally add the mean.	
 	17333.9735 <small>360 RPN</small>

This result means that ten percent of the households have monthly incomes greater than £17333.

## Factorial

The factorial key (!) is a secondary function on the up arrow key



The factorial function computes the product of the integers from 1 to x. Factorials can be useful when determining the maximum number of combinations or permutations.

For example – if you wanted to take photographs for your web site of your product line, how many different ways could the photographer arrange the five different products?

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

Keystrokes	Display
5 h !	1200000 360 RPN

This result confirms there are 120 different ways to arrange the five products for photography.

## Percent of sum %Σ

The percent of sum function permits you to compute the percentage that several values are of a total, while leaving the total intact. It computes the percentage the number in the X register is of the value for Σx

### Note



Assume that the keys A, B and C are NOT being used by the current program.

Example – Bread is made up of 5.4 grams of salt, 172.8 grams of seeds, and 866.7 grams of flour. What is the percentage by weight of each ingredient in the bread, and what is the total weight of the bread (before cooking, off course).

Keystrokes	Display
<b>g</b> <b>CLΣ</b>	
<b>5.4</b> <b>STO</b> <b>A</b> <b>Σ+</b>	00000 <small>360 RPN</small>
<b>172.8</b> <b>STO</b> <b>B</b> <b>Σ+</b>	20000 <small>360 RPN</small>
<b>866.7</b> <b>STO</b> <b>C</b> <b>Σ+</b>	30000 <small>360 RPN</small>

### Note



We will use storage registers A, B and C to hold temporary copies of the three values – and the key stroke sequence above might feel a little confusing to start with when (for example) you observe that we are using the "A" key followed immediately by the same key (but as Σ+)

When the A key is used immediately after the STO key, the calculator is in temporary alpha mode – which means it will then directly accept the names of 9 from the 12 special stack registers (Y,A,B,C,D,I,L,J, and K). In this context when we press "A", the calculator knows to store the number into the "A" stack register. Once that command is completed, the calculator then switches out of temporary alpha mode and the next press of the A key will invoke the default command of Σ+

First figure out the percentage of salt (the weight is in register A)

Now that the data is accumulated in the calculator find the percentage of each as follows	
<b>RCL</b> <b>A</b>	54000 <small>360 RPN</small>
<b>h</b> <b>STAT</b> <b>%</b> <b>↗</b> <b>%Σ</b> <b>XEQ</b>	05.168 <small>360 RPN</small>

The percentage of salt in the bread is just over half a percent.

### Note



Observe that when selecting the catalogue function %Σ, we quickly located the relevant part of the catalogue by typing f %

Now the percentage of seeds (weight in the B register)

RCL B	1728000
h STAT % $\frac{1}{2}$ $\% \Sigma$ XEQ	165375

The percentage of seeds in the bread is just over 16.5%. Now the percentage of flour (the weight is in the C register)

<div> <div>RCL</div> <div>C</div> </div>	<div>360 RPN</div> <div>866.7000</div>
<div> <div>h</div> <div>STAT</div> <div>%</div> <div><math>\frac{\square}{\square}</math></div> <div><math>\% \Sigma</math></div> <div>XEQ</div> </div>	<div>360 RPN</div> <div>829457</div>

The percentage of flour in the bread is nearly 83%. Summing the three as a check gives  $0.5168 + 16.5375 + 82.9457 = 100\%$  as we would expect.

### Percentage Difference $\Delta\%$

The percentage difference key (a secondary function of the minus key) gives you the percentage difference – that is, the relative increase or decrease between two numbers. To find the percent difference

1. Key in the first number (typically the first number that occurs first in time)
2. Press ENTER
3. Key in the second number
4. Press **g**  $\Delta\%$

The formula used in this function is:-

$$\Delta\% = \frac{100 (x - y)}{y}$$

A positive result signifies an increase in value, whereas a negative result signifies a decrease. Consider the example where a vehicle is appraised by an insurance company in 2009 and valued at £4700. It is an appreciating asset and in 2012 the same vehicle is valued at £6500. By what percent did the value of the vehicle rise from 2009 to 2012?

4700	ENTER	360 RPN	47000000	360 RPN
6500	g	$\Delta\%$	382979	360 RPN

The vehicle has increased in value by 38.29%.

As a check note that  $4700 \times 13829 = 649963$

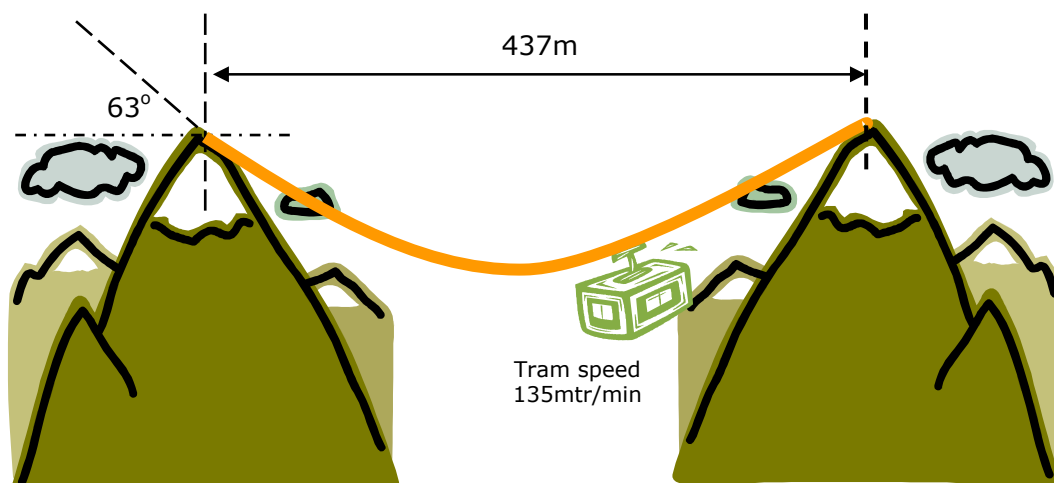
## Hyperbolic Functions

The WP 34S provides a set of six hyperbolic functions.

<b>f</b> <b>HYP</b> <b>SIN</b>	Hyperbolic sine
<b>f</b> <b>HYP</b> <b>COS</b>	Hyperbolic cosine
<b>f</b> <b>HYP</b> <b>TAN</b>	Hyperbolic tangent
<b>g</b> <b>HYP<sup>-1</sup></b> <b>SIN<sup>-1</sup></b>	Inverse hyperbolic sine
<b>g</b> <b>HYP<sup>-1</sup></b> <b>COS<sup>-1</sup></b>	Inverse hyperbolic cosine
<b>g</b> <b>HYP<sup>-1</sup></b> <b>TAN<sup>-1</sup></b>	Inverse hyperbolic tangent

Note how these functions employ three key strokes, and that you only have to press the f (or g) shift key one time in the key sequence. Hyperbolic functions are used in the same way as ordinary trigonometric functions with one important difference. Hyperbolic functions always assume that their input argument is expressed in radians, and that the inverse hyperbolic functions always provide a result in radians.

In a mountainous region of the Alps, a tram carries tourists between two peaks that are the same height and 437 metres apart. How long does it take the tram to travel from one peak to another if it moves at 135 meters per minute? Before the tram latches onto the cable, the angle from the horizontal to the cable at its point of attachment is found to be  $63^\circ$



The travel time is given by the formula

$$t = \frac{437 \tan(63^\circ)}{135 \sinh^{-1}(\tan(63^\circ))}$$

<b>g</b> <b>DEG</b>	360 RPN 00000
<b>437</b> <b>ENTER</b>	360 RPN 4370000
<b>63</b> <b>f</b> <b>TAN</b> <b>ENTER</b>	360 RPN 19626
Note how using the ENTER key after pressing TAN places two copies of tan(63) in the stack. We need this value for both the numerator and denominator.	
<b>g</b> <b>HYP<sup>-1</sup></b> <b>SIN<sup>-1</sup></b>	360 RPN 14268
<b>135</b> <b>*</b>	360 RPN 1926164
<b>÷</b>	360 RPN 00102
<b>*</b>	360 RPN 44527

Travel time between the two peaks will be marginally under 4½ mins.

To show this slightly more accurately in a format of hours, minutes and seconds, divide by 60 (to convert it from minutes into seconds) and then switch to the H.MS display format using the following key strokes

<b>60</b> <b>÷</b>	360 RPN 00742
<b>f</b> <b>H.MS</b>	360 0° 4' 27.16"

The display confirms that at a constant speed, the tram will require 4 minutes and 27.16 seconds to get from peak to peak.

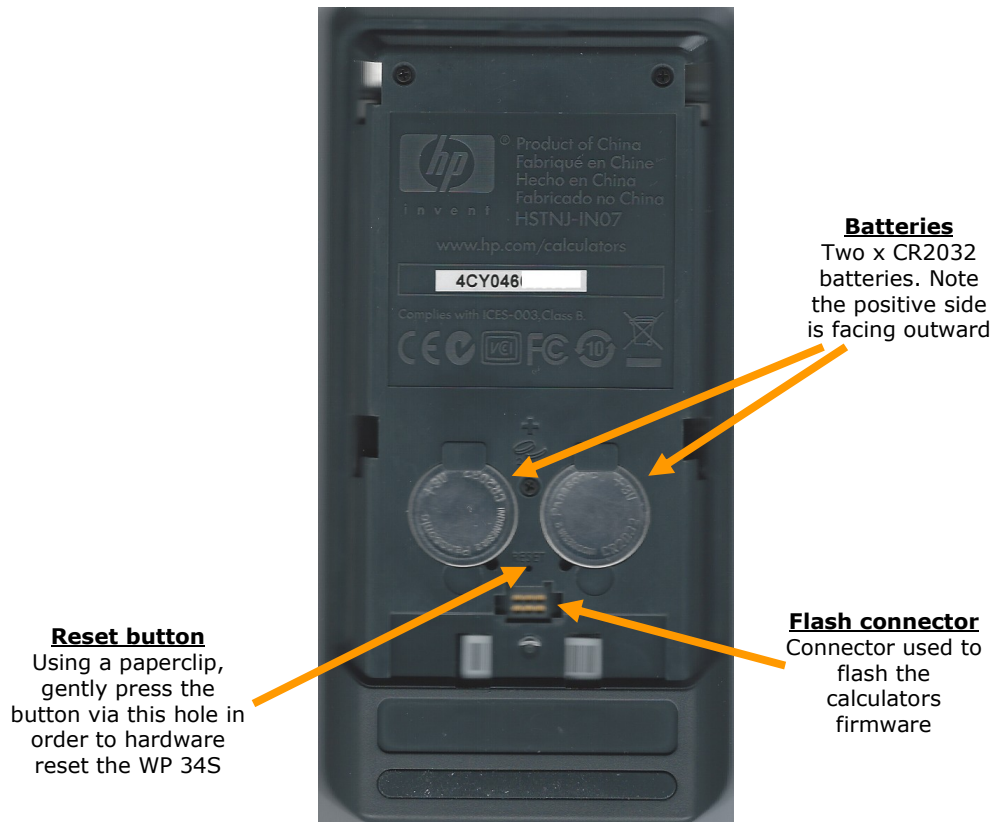
A large teal-colored rounded rectangle with rounded corners, centered on the page. It contains the title text in white.

# Part 10

## WP 34S Hardware & Firmware

## Calculator Hardware

The calculator has a rear cover that can be removed by gently pushing upwards to reveal the battery compartment.



### Replacing the batteries

The calculator uses two 3 volt CR2032 lithium batteries. A low battery symbol appears when the battery power is nearly exhausted (you can read the battery voltage by using the BATT command in the **X.FCN** catalogue – which will return the voltage in the X register – typically 2.8 to 3volts). Use only fresh batteries when replacing the battery and do not use rechargeable batteries.

To install a new pair of batteries

1. Turn the calculator off (leaving it on while removing the batteries may cause the loss of all memory contents)
2. Remove the back cover
3. Remove ONE battery (either left or right) and replace with a new CR2032 battery with the positive side facing outwards.
4. Remove the second battery and replace with a new CR2032 battery with the positive side facing outwards.
5. Replace the back cover.

### Warning

**There is a danger of explosion if the batteries are replaced incorrectly. Replace ONLY with the same or equivalent type recommended by the manufacturer. Dispose of used batteries according to the manufacturer guidelines. Do not mutilate, puncture or dispose of batteries in fire. The batteries can burst or explode releasing hazardous chemicals.**



## Hardware reset

When the calculator is reset by pressing the small push button at the back of the machine (see above picture) it goes through a sequence of two steps.

1. It clears everything – all memory, all variables are cleared.
2. It restores its memory using the contents of the current backup stored in the calculators flash RAM.

### Note



*Step 2 is worthy of note here because when the machine is first purchased, the backup area of flash RAM will be effectively clear of any data, and it will remain clear until the user creates a backup of the contents of the machine using either the ON & STO key combination, or the SAVE command in the P.FCN catalogue (both described in sections above).*

*If a backup of the machine has never been made, then unfortunately a hardware reset will cause the restoration of an empty backup and so will result in the loss of all calculator memory.*

To reset the device use the following steps.

1. Open the back cover
2. Using a paperclip or something similar, gently push into the plastic hole marked reset (see picture above). Be very gentle as little or no pressure is required.
3. Replace the back cover.

When you next turn the calculator on, the screen will remain blank for a short 2 second delay while the entire contents of flash RAM are copied into main memory.

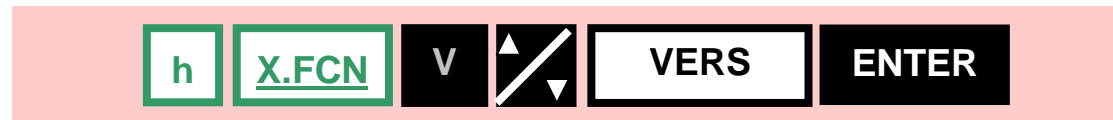
After that, the calculator display will announce the word "Restored".

## Calculator Firmware

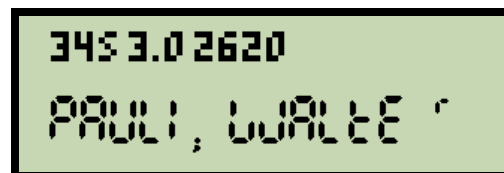
The software for the WP 34S is a work in progress, and at the time of writing is changing almost on a daily basis as features are adjusted and bugs fixed. You will at some point need to establish what version of software you are using.

### *What version firmware is my calculator using?*

To find out the version of firmware on your WP 34S use the VERS command in the X.FCN catalogue. Use the following keystrokes.



As an example - a calculator running build 2620 of version 3.0 of the firmware will show the following display. Note that the display includes a much deserved credit to both Paul Dale and Walter Bonin – two key originators in the development of both the hardware and firmware).



### *Upgrading firmware*

In order to upgrade the firmware on the WP 34S, a special cable is required which mates with the 6 pin connector under the battery cover (see the flash connector in the diagram above). At the time of writing (May 2012) these cables are still available but supplies are limited. Developments are being undertaken to explore an alternate connection mechanism using a micro USB interface but this is embryonic at this time and would involve retrofitting a micro USB to an existing machine not already fitted.

A small number of web based companies offer a WP 34S re-flashing service with postage costs to be borne by the customer.

#### Note



*During the draft of this guide I used three test calculators, and flash upgraded all three umpteen times. Aside from the fact it fixes a great many bugs in one stroke, it is also an extremely safe process (I haven't heard of a single instance of a user converting their machine into a brick after flash upgrading the firmware). Keeping that veiled warning in mind, if you live in the UK and are stuck trying to get your machine upgraded drop me a line and we may be able to work something out. All I'd require is that you arrange and pay for the post both ways.*

# Subject Index

## I

ICOMPL · 67, 68

## 2

2COMPL · 67, 68

## A

A · 2, 10, 17, 20, 22, 31, 33, 35, 52, 53, 54, 57, 58, 59, 60, 61, 62, 63, 67, 75, 87, 88, 89, 103, 104, 105, 106, 107, 108, 109, 110, 113, 117, 118, 122, 124, 125, 131, 134, 135, 136, 137, 141, 143, 147, 167, 175, 176, 180, 182  
 absolute value · 82, 147  
 add · 2, 44, 45, 47, 49, 61, 76, 100, 117, 127, 128, 135, 151, 153, 154, 168, 173  
 Addition · 21, 78, 79, 83  
 ALL · 10, 24, 28, 29  
 alpha · 10, 17, 34, 35, 52, 53, 60, 62, 63, 66, 98, 100, 103, 104, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 134, 135, 136, 137  
 Alpha · 10, 17, 34, 35, 52, 53, 60, 62, 63, 66, 98, 100, 103, 104, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 134, 135, 136, 137, 163  
 AND · 52, 55, 83, 84, 90, 92, 104  
 angular format · 31  
 annunciator · 74, 75, 76, 80, 82, 117, 126, 127, 128, 129, 134, 135, 144, 145, 147, 148, 151  
 approximate equality · 61  
 Archiving · 130  
 Arithmetic Shift · 89  
 Arrows · 122  
 ASR · 89  
 Automatic Memory Stack · 38

## B

B · 2, 19, 22, 35, 52, 53, 59, 60, 61, 62, 63, 67, 75, 77, 81, 103, 104, 105, 107, 108, 113, 117, 124, 125, 135, 141, 151, 155, 161, 176  
 Backing up · 58, 59  
 base · 2, 12, 14, 19, 65, 67, 71, 73, 76, 77, 78, 81, 82, 83, 88, 89, 92, 93, 94, 95, 158  
 BASE · 66, 67, 71  
 batteries · 180  
 BC? · 91  
 BEG · 12  
 binary · 65, 67, 68, 71, 72, 73, 80, 87, 88, 89, 92, 93, 94, 95, 158  
 Bit Shifting · 87  
 Bit wise operations · 75  
 Branching · 103

BS? · 91  
 byte · 73

## C

C · 2, 14, 22, 42, 45, 51, 52, 53, 58, 59, 60, 61, 62, 63, 75, 97, 100, 103, 104, 105, 107, 108, 113, 117, 124, 125, 135, 139, 141, 151, 153, 156, 168, 176, 182  
 Calling · 103  
 carry · 75, 76, 78, 79, 80, 81, 82, 87, 88, 89, 90  
 Cartesian · 152, 153  
 Catalogue · 2, 28, 33, 34, 35, 36, 37, 42, 53, 55, 58, 61, 66, 67, 68, 71, 73, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 105, 114, 118, 125, 130, 137, 138, 140, 145, 148, 149, 152, 159, 161, 163, 170, 180, 182  
 Catalogues · 33, 118  
 CB · 91  
 CF · 75, 77  
 Chain Arithmetic · 45  
 Chain calculations · 23  
 CIR · 98, 101, 103, 104, 105, 106, 107, 108, 110  
 circumference · 97, 98, 101, 103, 104, 105, 108, 110, 112, 115  
 Clα · 125  
 CLΣ · 56  
 CLALL · 56  
 clear bit · 91  
 Clearing · 18, 42, 58  
 Clearing the Stack · 42  
 CLSTK · 42  
 CLx · 43  
 complement modes · 75  
 Complex mode · 151, 152  
 complex stack · 154  
 compress · 126, 127, 129, 134  
 CONST · 33, 35, 163  
 Constant Arithmetic · 48  
 constants · 10, 15, 106, 163  
 CONV · 33, 159, 161  
 conversions · 2, 15, 31, 33, 144, 145, 158, 159, 160  
 Correlation Coefficient · 170  
 CPX · 2, 33, 34, 66, 108, 109, 110, 121, 151, 152, 153, 155, 163  
 CR2032 · 180  
 crystal · 14, 33  
 cumulative statistics · 52, 55

## D

D · 2, 22, 34, 36, 52, 53, 59, 60, 61, 62, 63, 75, 103, 104, 105, 107, 108, 110, 113, 117, 124, 125, 135  
 D.MMYYYY · 36  
 date · 33, 34, 36, 37  
 DATE · 37  
 DBL/ · 10, 92, 94, 95  
 DBLR · 92, 95  
 DBLx · 92, 93

decimal · 18, 24, 26, 27, 28, 29, 30, 34, 35, 36, 37,  
62, 63, 65, 66, 67, 75, 97, 110, 143, 148, 158  
decimal representation · 75  
DECM · 65  
deemphasised four bits · 73  
DEG · 2  
degree · 13, 31, 137, 170  
degrees · 19, 31, 158, 159  
delete · 97, 102, 108, 109, 110, 114  
DENANY · 148, 149  
DENFAC · 148, 149  
DENFIX · 145, 146, 148, 149  
DENMAX · 145, 146, 147, 148, 149  
Denominator · 144  
Direct · 54, 111  
DISP · 28  
display contrast · 18  
display format · 24, 148, 178  
divide · 49, 78, 82, 89, 92, 115, 151, 154, 178  
Division · 78, 82  
Double · 92, 93, 94, 95  
drops · 45, 47, 48, 83, 94  
DSE · 136, 137

## E

edit · 97, 110, 131  
EEX · 2, 31, 32, 50, 67, 108  
END · 98, 101, 106  
ENG · 24, 26, 28, 29  
Engineering / Scientific Override · 27, 29  
Engineering display notation · 26  
ENGOVR · 27, 29  
ENTER · 21, 23, 32, 36, 42, 43, 44, 45, 46, 48, 49,  
50, 51, 60, 61, 62, 63, 82, 93, 95, 100, 108, 118,  
124, 127, 131, 135, 153, 155, 156, 160, 163, 165,  
166, 167, 169, 176, 178  
Exact fraction · 144  
EXIT · 2, 12, 17, 18, 34, 56, 57, 59, 63, 108, 118,  
133  
Exponents · 31

## F

Factorial · 10, 174  
factors · 149  
Firmware · 182  
FIX · 24, 27, 28, 29, 35, 62  
Fixed display notation · 27, 29  
flags · 52, 57, 75, 81, 138, 139, 147, 153  
flash memory · 58  
fractions in programs · 149

## G

global · 52, 58, 98, 100, 103, 104, 105, 107, 108,  
114, 126, 134  
Global registers · 52, 53  
GRAD · 2  
gradian · 31  
gradians · 31, 158  
Greek · 10, 35, 118, 124, 163

Gt · 145, 147, 148  
GTO · 102, 103, 105, 108, 110, 111, 114, 131, 134,  
136, 137

## H

H.d · 2, 12, 143  
H.MMSS · 36  
Hardware · 180, 181  
Crystal · 14, 33  
hexadecimal · 12, 65, 66, 67, 77, 78, 92, 158  
horizontal scroll · 30, 69, 148  
hotkey labels · 107, 135  
hotkey local labels · 104, 107, 110  
Hyperbolic · 10, 177

## I

I · 10, 12, 13, 49, 51, 53, 59, 60, 61, 62, 63, 73, 75,  
100, 113, 124, 125, 135  
IBASE? · 67  
improper fraction · 143, 144, 145  
Indirect · 54, 55, 112, 113  
Indirect program execution · 112  
Inexact fraction · 145  
Integer · 65, 77, 144  
Modes · 74  
Intermediate results · 23  
INTM? · 67  
invert · 83, 161, 173

## J

J · 10, 53, 59, 60, 61, 62, 63, 75, 113, 124, 125, 135  
Justification · 88

## K

K · 10, 53, 59, 60, 61, 62, 63, 75, 113, 124, 135  
Keystroke Programming · 97

## L

L · 2, 10, 35, 50, 51, 53, 59, 60, 61, 62, 63, 75, 113,  
124, 125, 135, 168  
label · 2, 45, 98, 100, 103, 105, 106, 107, 108, 109,  
110, 111, 112, 113, 114, 126, 134, 135, 151, 155,  
156  
Larger fractions · 147  
LAST X · 50, 51  
LBL · 98, 100, 105, 134, 136  
least squares · 168, 169  
lifted · 45, 46, 166, 168  
Linear Estimation · 170  
Linear Regression · 168  
LJ · 88  
local · 57, 60, 61, 62, 103, 104, 105, 106, 107, 108,  
109, 110, 111, 135, 138, 139, 140, 141, 151  
Local Data · 57, 138, 139

locating catalogue functions · 35  
 LocR · 60, 61, 62, 138, 139, 140  
 Logic symbols · 123  
 Logical operations · 83  
 Logical Shifts · 87  
 long number displays · 69, 71  
 Loop Control · 136  
 LST X · 50  
 Lt · 145, 147  
 LZOFF · 73  
 LZON · 73

## M

mantissa · 14, 30, 67, 147, 148  
 Mantissa · 30, 67, 68, 148  
 MASKL · 92  
 MASKR · 92  
 math symbol · 118, 119, 121, 122, 123  
MATRIX · 33  
 max precision · 149  
 mean · 148, 165, 166, 168, 171, 172, 173  
 Memory  
   Access · 54, 55, 112, 113  
 Memory Organisation · 52  
 MIRROR · 92  
 Mixed fraction · 143  
MODE · 2, 28, 33, 34, 35, 36, 53, 55, 66, 67, 68, 71,  
   73, 89, 145, 148, 149  
 Multiplication · 78  
 multiply · 21, 23, 31, 48, 49, 50, 60, 61, 63, 77, 92,  
   98, 105, 111, 135, 151, 154

## N

NAND · 83, 84  
 NOR · 83, 85  
 Normal Distribution · 171  
 NOT · 80, 82, 83, 107  
 Number labels · 110, 112  
 Numerator · 144

## O

octal · 65, 67, 78, 92, 158  
 One number functions · 19  
 OR · 24, 34, 55, 83, 84, 85, 126  
 Order of Execution · 49  
 Out Of Range Error · 55  
 overflow · 74, 75, 77, 78, 81, 82  
 Overflow · 75, 81  
 Overflow flag · 75

## P

P.FCN · 33, 42, 58, 114, 125, 130, 137, 138  
 P/R · 102, 105, 108, 110  
 parallel · 115, 128, 130, 131  
 parenthesis · 13, 49  
 Percent of sum · 175  
 Percentage Difference · 176

Preferences · 10, 12, 14, 18, 19, 23, 29, 33, 34, 36,  
   37, 39, 40, 47, 48, 51, 52, 56, 59, 101, 102, 108,  
   110, 115, 125, 126, 127, 129, 130, 131, 134, 135,  
   138, 139, 140, 147, 154, 165, 169, 176, 177, 178,  
   182  
 PROB · 33  
 Program  
   Labels · 2, 10, 14, 17, 19, 20, 22, 31, 33, 34, 35,  
     36, 42, 45, 51, 52, 53, 54, 57, 58, 59, 60, 61,  
     62, 63, 67, 75, 77, 81, 87, 88, 89, 97, 100, 103,  
     104, 105, 106, 107, 108, 109, 110, 113, 117,  
     118, 122, 124, 125, 131, 134, 135, 136, 137,  
     139, 141, 143, 147, 151, 153, 155, 156, 161,  
     167, 168, 175, 176, 180, 182  
 Program Input / Output · 115  
 program mode · 100, 105, 106, 108, 109, 110, 111,  
   126, 131, 134, 137, 151  
 Program-Entry · 102  
 Programming · 97  
 Program-Step · 102  
 PROMPT · 125, 127, 128, 129, 130, 132  
 proper fraction · 143, 144, 145, 147  
 PSE · 125, 126, 128  
 Punctuation · 120  
 Punctuation symbol · 120

## R

R/S · 101, 108, 114, 129, 130, 131, 132  
 RAD · 2  
 radian · 31  
 radians · 31, 158, 177  
 radius · 97, 98, 101, 115  
 RAM · 10, 52, 58, 59, 104, 114, 130, 181  
 RCL · 10, 12, 50, 51, 52, 53, 55, 59, 60, 61, 62, 108,  
   125, 138  
 reciprocals · 22  
 Register  
   A · 2, 10, 17, 20, 22, 31, 33, 35, 52, 53, 54, 57,  
     58, 59, 60, 61, 62, 63, 67, 75, 87, 88, 89, 103,  
     104, 105, 106, 107, 108, 109, 110, 113, 117,  
     118, 122, 124, 125, 131, 134, 135, 136, 137,  
     141, 143, 147, 167, 175, 176, 180, 182  
   B · 2, 19, 22, 35, 52, 53, 59, 60, 61, 62, 63, 67,  
     75, 77, 81, 103, 104, 105, 107, 108, 113, 117,  
     124, 125, 135, 141, 151, 155, 161, 176  
   C · 2, 14, 22, 42, 45, 51, 52, 53, 58, 59, 60, 61,  
     62, 63, 75, 97, 100, 103, 104, 105, 107, 108,  
     113, 117, 124, 125, 135, 139, 141, 151, 153,  
     156, 168, 176  
   D · 2, 22, 34, 36, 52, 53, 59, 60, 61, 62, 63, 75,  
     103, 104, 105, 107, 108, 110, 113, 117, 124,  
     125, 135  
   L · 50, 51  
   T · 39, 40, 42, 47, 48  
   X · 2, 12, 18, 19, 27, 36, 38, 40, 42, 43, 47, 48,  
     50, 54, 60, 62, 66, 67, 68, 73, 74, 78, 82, 83,  
     87, 88, 89, 90, 91, 92, 93, 94, 97, 98, 104, 105,  
     111, 112, 125, 127, 128, 129, 132, 133, 134,  
     135, 137, 138, 140, 141, 146, 149, 153, 158,  
     159, 160, 163, 165, 166, 167, 168, 173, 175,  
     180  
   Y · 41, 42, 46, 47, 48, 79, 83, 88, 93, 95, 112,  
     113, 115, 128, 153, 154, 163, 165, 166, 167,  
     168

Z · 42, 48, 60, 94, 95  
 REGS · 53, 55, 58  
 Remainder · 82, 95  
 RESET · 56  
 Restored · 58, 181  
 Restoring a backup · 59  
 RJ · 88  
 RL · 89, 90  
 RLC · 89, 90  
 RMDR · 82, 95  
 Roll · 39, 40  
 roll down · 39, 88, 93, 100  
 roll up · 39  
 Roll up · 40  
 roots · 22  
 Rotating · 87, 89  
 Rotation · 90  
 Rounding · 147  
 RR · 89, 90  
 RRC · 89, 90  
 RTN · 57, 103, 140, 141  
 run · 2, 58, 97, 98, 101, 105, 108, 110, 126, 129,  
 130, 131, 132, 134, 137, 148  
 run mode · 98  
 run/stop · 2, 129, 130, 132

---

## S

SB · 91  
 SBR · 52, 139  
 SCI · 24, 28, 29  
 Scientific display notation · 24  
 SCIOVR · 27, 29  
 set bit · 91  
 SETDAT · 36, 37  
 SETTIM · 10, 36  
 Setting Preferences · 18  
 SETUK · 34  
 SF · 75  
 Sigma Data · 55  
 sigma data entry · 22  
 sign of the number · 18  
 SIGNMT · 67, 68  
 SL · 87  
 SMODE? · 67  
 special characters · 118  
 SR · 87, 89  
 Stack · 38, 39, 42, 44, 46, 47, 57, 113  
 Stack Arithmetic · 44  
 Stack control · 39  
 stack drop · 45, 47, 48, 83, 94  
 stack lift · 47, 167  
 Standard Deviation · 166  
 STAT · 33, 170  
 Statistical functions · 10  
 STATUS · 56, 57, 58, 139  
 Status and configuration data · 52  
 STO · 10, 52, 53, 55, 59, 60, 61, 62, 104, 108, 138  
 Subroutine · 57  
 subtract · 49, 151, 154  
 Subtraction · 78, 79  
 SUMS · 33  
 Switching On · 17  
 Symbols · 118, 119, 120, 122

---

## T

T · 38, 39, 40, 42, 46, 47, 48, 52, 53, 59, 60, 61, 62,  
 63, 75, 100, 113, 124, 125, 135, 154  
 T register · 39, 40, 42, 47, 48  
 Temporary Alpha Layout · 63  
 temporary alpha mode · 10, 60, 62, 125, 135  
 TEST · 33, 53, 61, 67, 91, 105, 118, 138, 140  
 test commands · 61  
 Testing bits · 91  
 time · 12, 14, 18, 19, 23, 29, 33, 34, 36, 37, 39, 40,  
 47, 48, 51, 52, 56, 59, 101, 102, 108, 110, 115,  
 125, 126, 127, 129, 130, 131, 134, 135, 138, 139,  
 140, 147, 154, 165, 169, 176, 177, 178, 182  
 TIME · 37  
 true population · 167, 168  
 truncated · 36, 37, 147, 148  
 Two-number functions · 21

---

## U

UK · 12, 18, 33, 34, 169  
 UK mode · 18  
 Unit Conversions · 31  
 UNSIGN · 67, 68, 89  
 unsigned mode · 67, 68, 74, 79, 82, 89  
 Upgrading firmware · 182

---

## V

VERS · 182  
 version firmware · 182  
 VIEW · 125  
 VWα+ · 125

---

## W

WSIZE · 67  
 WSIZE? · 67

---

## X

X · 2, 12, 18, 19, 21, 23, 27, 33, 35, 36, 37, 38, 39,  
 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,  
 53, 54, 57, 59, 60, 61, 62, 63, 66, 67, 68, 69, 73,  
 74, 75, 78, 79, 82, 83, 84, 85, 86, 87, 88, 89, 90,  
 91, 92, 93, 94, 97, 98, 104, 105, 108, 111, 112,  
 113, 115, 124, 125, 127, 128, 129, 131, 132, 133,  
 134, 135, 137, 138, 140, 141, 146, 149, 152, 153,  
 154, 158, 159, 160, 161, 163, 165, 166, 167, 168,  
 169, 170, 171, 172, 173, 175, 180, 182  
 X exchange Y · 39, 41, 69, 153  
 X register · 2, 12, 18, 19, 27, 36, 38, 40, 42, 43, 47,  
 48, 50, 54, 60, 62, 66, 67, 68, 73, 74, 78, 82, 83,  
 87, 88, 89, 90, 91, 92, 93, 94, 97, 98, 104, 105,  
 111, 112, 125, 127, 128, 129, 132, 133, 134, 135,  
 137, 138, 140, 141, 146, 149, 153, 158, 159, 160,  
 163, 165, 166, 167, 168, 173, 175, 180  
 X.FCN · 33, 35, 37, 83, 84, 85, 86, 87, 88, 89, 90,  
 91, 92, 152, 180, 182

XEQ · 2, 34, 101, 102, 103, 107, 108, 111, 160  
XNOR · 83, 86  
XOR · 83, 85, 86  
XTAL? · 33

---

## ***Y***

Y · 21, 38, 39, 41, 42, 45, 46, 47, 48, 52, 53, 57, 59,  
60, 61, 62, 63, 69, 75, 79, 82, 83, 88, 93, 94, 95,  
108, 112, 113, 114, 115, 124, 125, 128, 131, 133,  
135, 152, 153, 154, 163, 165, 166, 167, 168, 169,  
170

Y register · 41, 42, 46, 47, 48, 79, 83, 88, 93, 95,  
112, 113, 115, 128, 153, 154, 163, 165, 166, 167,  
168  
 $y=mx+c$  · 168, 169  
 $y^x$  · 21, 154

---

## ***Z***

Z · 38, 42, 46, 47, 48, 52, 53, 59, 60, 61, 62, 63, 75,  
94, 95, 113, 124, 125, 135, 154, 155  
Z register · 42, 48, 60, 94, 95

