

Programming Notes and Hints for the HP 35s Scientific Calculator

Frederick Ruland

**Programming Notes and Hints for the
HP 35s Scientific Calculator
By
Frederick S. Ruland**

Copyright © 2017 Frederick S. Ruland

All rights reserved

This book is dedicated to the memory of Marian Rejewski, Jerzy Rozycki, and Henryk Zygalski, the Polish mathematicians who broke the German Enigma code.

Table of Contents:

1.0 Quick Start Section 1

1.1 On and Off 1

1.2 Programming – A Quick Start Example 1

1.3 Running the Quick Start Example 2

1.4 Notes for the Quick Start Section 3

2.0 The Stack and the Display 3

2.1 Understanding the Stack..... 4

2.2 Entering Numbers..... 5

2.3 Operations Where Both the X and Y Appear on the Keys 6

3.0 Memory for Programs and Data 7

4.0 A Bit about Reverse Polish Notation (RPN)..... 7


5.0 Storing (STO) and Recalling (RCL) Numbers..... 8

6.0 Clearing Operations 9

7.0 Memory Menu..... 10

7.1 The Checksum..... 12

7.2 Deleting Programs from Memory..... 13






8.0 RCL(recall) and  VIEW 13


9.0 Editing Programs and More Details about the Commands 14

9.1 Let’s edit program B..... 16

9.2 Let’s add the two lines 16

10.0 Before writing a more useful program, let’s look at some
commands 18

10.1 RTN (Return) Statement	18
10.2 GTO (Go To) Statement	18
10.3 A Bit More About the GTO Statement	19
10.4 GTO (Followed and Not Followed by a Period).....	20
10.4.1 When NOT in Programming Mode	20
10.4.2 When IN Programming Mode.....	20
11.0 XEQ Statement (and an Additional Function of the RTN Statement)	21
11.1 An Aside Regarding GTO and XEQ	22
12.0 STO (Store) Statement	23
13.0 A Bit More About the RCL (Recall) Statement	24
14.0  X?0 (Comparing the X Stack Register with Zero)	24
14.1 X?0 Example.....	25
14.2 Let's Run the X?0 Program.....	27
15.0 Looping Capabilities.....	28
15.1  ISG (Increment and Skip if Greater Than)	28
15.2 Small Example to Shed Light on the  ISG Command	29
15.3 What You Should See When Executing the ISG Program	30
15.4 Let's Run the ISG Program	31
16.0 Guessing Game	32
17.0  X?Y (Comparing the X Stack Register With the Y Stack Register).....	34
17.1 Updated Guessing Game with  X?Y	35
18.0 Simple Program with a Few Commands Not Yet Used (Including Storage Arithmetic).....	36

19.0 More Looping capabilities: The  DSE (Decrement, Skip [next command] if Equal or Less Than)	38
19.1 DSE Program to Loop Five Times	39
19.2 Let's Run the DSE Program	40
20.0 Addressing Direct and Indirect Variables - The Functions (I) and (J)	41
20.1 Use of variables I and J and the functions (I) and (J)	41
20.2 Example Referencing Direct Variables	41
20.3 Example Addressing Indirect Variables.....	42
20.3.1 Indirect Variable Storage Requirements.....	43
20.4 Small Program to Input Data into Indirect Variables	44
20.5 Small Program to Read Data from Indirect Variables	45
20.6 Let's Run the Above Program to Read Data from the Indirect Variables	46
20.7 To Erase/Clear Indirect Registers	47
20.8 Clearing the 0 Indirect Register	47
21.0 Addition to the Guessing Game (Section 16.0) using RAND and INTG	48
22.0 Equation Solver (When the Equations Are Stored In the List) 50	
22.1 Entering an Equation	50
22.2 Three Approaches When Evaluating Equations.....	51
22.2.1 Method 1. EQN followed by ENTER.....	51
22.2.2 Method 2. EQN followed by SOLVE	53
22.2.3 Method 3. EQN followed by XEQ.....	54
23.0 Equation Solver (When the Equations Are NOT Stored in the Equation List)	56

23.1 FN = Program Label..... 56

23.2 How the 'FN =' Works Outside the Programming Mode 58

24.0 FLAGS 59

24.1 Flags Menu..... 59

24.2 FLAGS 0 through 4 60

24.3 FLAGS 5 through 9 61

24.4 FLAGS: 10 and 11 61

24.4.1 With Flag 10 set; Flag 11 set; With 'SOLVE = A' statement in the program 62

24.4.2 With Flag 10 set; Flag 11 set; Without the 'SOLVE = A' statement in the program..... 62

24.4.3 With Flag 10 clear; Flag 11 clear; With 'SOLVE = A' statement in the program 62

24.4.4 With Flag 10 clear; Flag 11 clear; Without 'SOLVE = A' statement in the program..... 62

24.4.5 With Flag 10 clear; Flag 11 set; With 'SOLVE = A' statement in the program 63

24.4.6 With Flag 10 clear; Flag 11 set; Without 'SOLVE = A' statement in the program..... 63

25.0 Displaying Messages in Programs (FLAG 10 and EQN) 64

26.0 Equivalent Traditional Programming Commands 66

26.1 Assignment Statements 66

26.2 COMMENTS 66

26.3 ARRAYS and INDICIES 67


26.4 IF STATEMENTS 67

26.5 DO LOOP 67


26.6 Go To 67

26.7 WRITE / PUT / PRINT 68

27.0 HP 35s ‘Do Loop’ Approach Example..... 68

28.0 IF THEN ELSE Equivalents (Using the  X?0 options) 69

29.0 Somewhat ‘Mysterious’ Commands..... 70

29.1 Pause:  PSE (pause)..... 70

29.2 RTN and STOP 70

29.3 Accessing the Stack Registers 71

29.3.1 Accessing the Stack Registers (Via the Equation List)..... 71

29.3.2 Accessing the Stack Registers (Via Programming Mode) 72

29.4 LASTX -- Last value of the X stack register (and something interesting that happens) 73

29.5 UNDO 74

30.0 Miscellaneous 75

30.1 Unlabeled Program Stored Above the Other Programs 75

30.2 To ‘Grab’ Stack Contents and Store As a Variable..... 76

30.3 Comment on Evaluating Equations..... 77

30.4 How to Step through a Program..... 78

30.5 The Integer Function Key..... 79

30.6 “Culled” Material I Didn’t Have the Heart to Cull..... 80

30.6.1 Relationship between the XEQ statement and the RTN statement 80

30.6.2 To Bring the T Stack Register into the X Stack Register 81

30.6.3 When Entering Numbers into an Executing Program..... 82

30.6.4 The Stack is the Key to Efficiency..... 82

31.0 Conditional Statement Summary 82
32.0 A Peculiarity about the Stack and the Recall (RCL)..... 83
33.0 Suggestion 84

Foreword

This little book is an attempt to supplement the HP 35s User's Guide for those things this author had trouble understanding or discovered only after trial and error. It concentrates on topics such as programming, using memory, using Reverse Polish Notation (RPN), using stack registers, program construction, and working with equations (the equation solver).

The book is a general programming guide, rather than a how-to guide for specific specialty areas. It does not discuss integration, mathematical transformations, logical operations, or conversion factors. It assumes RPN (Reverse Polish Notation) throughout.


Although the HP 35s lacks a computer interface and a graphics display, it is reasonably priced (about \$50 in June 2017), and has other desirable features. Its greatest strength is perhaps its capacity to store custom programs and equations specific to an individual's needs.


This book does not cover every aspect of the HP 35s. Therefore it is important to have the HP 35s User's Guide which is available free online via the following link:

<http://support.hp.com/us-en/product/hp-35s-scientific-calculator/3442983/manuals>

1.0 Quick Start Section

1.1 On and Off

Pressing  ON or simply pressing C turns the calculator on.


Pressing  OFF turns the calculator off.

1.2 Programming – A Quick Start Example


Here is a very simple (but admittedly not very useful) program, just to give you a sense of some basics. The program asks for the value of D, and then displays that value. Later there will be more complete and useful examples.


Let's assume you are in the non-programming mode where just two lines of numbers appear in the display.

GTO . . . Pressing GTO followed by two decimal points, sets a pointer to the top of the program memory.


 PRGM Takes you into programming mode. (You will see PRGM TOP)


Note: The line numbers below are automatically added using the program label as a prefix. Also, you don't have to move to the next line to type another command. It will jump to the next line when you start typing.

A001  LBL A Gives the program a label (i.e., a name), in this case 'A'.

A002  INPUT D Lets you enter a value for variable D. (You will be prompted to enter the value when the program runs.)

A003  VIEW D Displays the value of variable D.

A004  RTN Marks the end of the program, i.e., a return command.

Pressing C or pressing  PRGM will take you out of programming mode.

Note: The letters A and D above are in red on the keys.

1.3 Running the Quick Start Example

XEQ A ENTER Press XEQ then A then ENTER (Starts execution of program A.)

You will see the prompt “ D? ”

Type 1234 and press ENTER (Also see note below.)

Press R/S to continue execution. R/S is the Run / Stop command.

You will see “ D= 1,234 ”

Press R/S to continue execution.



At this point our simple program ends with the display of the X and Y stack registers. Programming details will be presented later.

Note: Technically you can omit pressing ENTER in this situation and go directly to pressing R/S. A bit more about this will be said in Section 2.2.

1.4 Notes for the Quick Start Section


The letters A to Z, as well as being variable names are also used to label (i.e., name) your programs. The variable names and program labels are independent of each other. That is, you can have a program labeled A, and also a variable named A.

The Run/Stop command (R/S) is used to restart the program when it is stopped, as well as stop the program when it is running.

A number of commands require a colored 'pre-selector' key  or . However the red program labels and variable names are context sensitive and do not require a pre-selector key.

2.0 The Stack and the Display

The HP 35s has a four register stack. The registers are designated as X, Y, Z, and T. The display window shows the contents of the X register (in the lower line) and the Y register (in the upper line). Above the Y register (but unseen) is the Z register, and further above is the T register.


Using the  R↑ key, and the R↓ key you can scroll the contents of the stack to different registers. I was originally under the impression that the registers themselves scrolled. However, it is the content which scrolls through the registers. Therefore, whatever

value you scroll into the lower display is then described as being in the X stack register.

2.1 Understanding the Stack

In very general terms, as values are entered they proceed up through the stack registers: X to Y, Y to Z, and Z to T. As the contents are 'consumed', they drop down (T to Z, Z to Y). As the content of the T register (which has no register above it) drops to the Z register, the original content of the T register is replicated in the new T register.

Here is an example:

First, clear the stack by pressing  CLEAR 5 (The 5 is the menu option to clear the stack.) There will be more about the CLEAR menu in Section 6.0.

Type a 1 and press ENTER. Type a 2 and press ENTER. Type a 3 and press ENTER. Type a 4 but **DON'T** press ENTER.

Using the R↓ key, observe the contents of the registers by scrolling down.

Continue to scroll until 4 is again in the X stack register (lower window).

Now press the + key.

You will see a 7 (the sum of the X and Y stack registers, i.e. the 4 and the 3) appear in the X stack register. (The value in the Y stack register has basically been consumed.)

The 2 from the Z stack register drops, and is now in the Y stack register.

The 1 from the T stack register drops and is now in the Z stack register.

Now to see what is in the T stack register, use the R↓, to again scroll.

You will see that another 1 has appeared. The reason is that when the contents of the T stack register drops, the original content is replicated in the new T stack register.

Scroll down again (to make 7 appear in the X stack register).

The stack (reading up) is now: 7, 2, 1, 1

2.2 Entering Numbers

As you type a number, it appears in the X stack register. What happens after that depends on what you do next.

If you press ENTER, the number you typed is placed into both the X and the Y stack registers.

For example: Type a 7 and press ENTER. You will then see a 7 in both the X stack register (lower window display) and in the Y stack register (upper window display).

If you now press the + key, the contents of the X and Y stack registers are added and the sum (i.e., 14) is placed in the X stack register.

If you type a 7 and don't press ENTER, you will see 7 in the X stack register and 0 in the Y stack register. (Zero because you previously cleared the stack).

Now press +

This will add the contents of the X and Y stack registers, and the result of the operation (7 + 0) will appear in the X stack register.

Note: The + always adds the contents of the X and Y stack registers.

2.3 Operations Where Both the X and Y Appear on the Keys

You will notice that on some keys both Y and X appear, for example Y^X . (The displayed order on the key is important, that is, it does not show the reverse X^Y .)

The way to enter the required values is to first enter the Y and then the X. Take for example 2^3 . (So Y will be the 2 and X will be the 3.)

Note: This is where the duplication which occurs when ENTER is pressed is important.

So for the 2^3 calculation: Type 2 and press ENTER. This puts 2 into both the X stack register and more importantly also into the Y stack register. Now type 3 which — without pressing ENTER — will go only into the X stack register.

At this point, you should see a 3 in the X stack register, and a 2 in the Y stack register.

Now press the Y^X key, and you will see 8 appear in the X stack register. Since the operation 'consumes' the previous values in the

X and Y stack registers, the Y stack register will now show 0 (assuming the stack was previously cleared). There will be more about the stack a little later.

3.0 Memory for Programs and Data

You can save up to 26 programs using any letters A to Z, up to the limit of available memory (30 KB) for programs and data — specifically 30,192 bytes.

There are also 26 easily accessed locations for the storage of numbers. These locations (referred to as direct variables) are identified by the letters (i.e., variable names) A through Z. These letters are displayed in red on the keys. The use of these direct variables does not subtract from the available memory for programs and data.

There is no connection between the letters used for program names and the letters used for variable names. For example, you can have a program A and also a variable A.

There is also no connection between the variable names (A to Z) and the stack register names: X, Y, Z, and T. They are completely separate things.

There is information about checking the memory status in the Memory Menu Section 7.0.

4.0 A Bit about Reverse Polish Notation (RPN)

There is plenty of online information regarding the history of, and use of, RPN. Here is an overly simplified explanation of using RPN.

To divide 100 by 20:

The 100 is entered, then (rather than an operator being given), the 20 is entered. Only after the numbers are entered is the operator given.

Specifically:

100 Press ENTER

20 Don't press ENTER

÷ Press divided by


5 Displayed

Note: There is a handy key [X<>Y] which exchanges the contents of the X and Y stack registers, if you actually intended to divide 20 by 100.

Additional information about RPN will be interspersed as we go along.

5.0 Storing (STO) and Recalling (RCL) Numbers

For Example, to store the value 123 in variable A:

Type 123. then  **STO A**



You will also see 123 displayed in the X stack register (lower line in the display).

To confirm that 123 is stored in variable A, first clear the X stack register by pressing the C key.


Now to recall variable A, press RCL **A**

You will see 123 appear in the X stack register.

Check Section 8.0 and 13.0 for more information on the RCL command.


Note: There is a feature, a sort of combination of the STO and the RCL, activated by the  X  key, which exchanges the contents of the X stack register with the contents of any selected variable (direct or indirect). The display, after pressing the above keys, looks like X <> . The chosen variable is typed where the underscore appears. (See page 3.8 of HP 35 User's Guide.)

6.0 Clearing Operations


If you press  CLEAR you will see the CLEAR menu:


Use the v key (under the **MEM**) to scroll down.

1X	2VARS
3ALL	4Σ
5STK	6CLVARX

 CLEAR 1 Clears only the X stack register.


 CLEAR 2 Clears the 26 variable storage registers (A to Z).


 CLEAR 3 Clears the four stack registers (X, Y, Z, and T), the variables A to Z, and the stored programs (A to Z).

Note: After  CLEAR 3, you will see “CLR ALL? Y N”.

To clear all, use the < arrow to underline the Y, and press ENTER.


 CLEAR 4 Clears the 6 statistical registers: n Σx Σy Σx^2 Σy^2 Σxy

 CLEAR 5 Clears the 4 stack registers (X, Y, Z, and T).

 CLEAR 6 Clears the indirect registers. (More about this later.)

Before we get to any editing, you may want to see what programs you have stored in memory.

7.0 Memory Menu

If you press  MEM you will see something like the following:

1VAR	2PGM
0	30,168

Note: The 1 and 2 before 1VAR and 2PGM are the option numbers and not the number of variables and number of programs.

You can use the first option to check the values of the 26 memory variables (A to Z).

MEM 1 Press 1 for VAR

You might see something like the following:

A = 8.000 (i.e., variable A = 8)

Use the up and down arrows: ^ v to scroll the listing.

B = 2.000 (i.e., variable B = 2)

Press C to 'escape' from the list.

Use the second option to see the names (i.e., LBLs) of your stored programs and their length (LN).

MEM 2 Press 2 for PGM option.

You will see, for example:

LBL B Your program labeled B

LN = 12 Length of program in bytes.

Use the up and down arrows: ^ v to scroll the listing.

LBL A Your program labeled A

LN = 24 Length of program in bytes.


When you press **MEM**, in addition to seeing 1VAR and 2PGM, you will see two numbers in the lower row (the 0 and the 30, 168 above). The one on the left indicates the number of indirect

variables used (more about indirect variables in Section 20). The one on the right indicates the amount of available memory.

7.1 The Checksum

In addition to the LN mentioned above, there is an additional useful piece of information. This is the checksum. It is a four digit hexadecimal number that uniquely identifies a program. For example, if you delete a program and then retype it, or you enter a program from the literature, the checksum gives you a way to check on the accuracy of your entry.

To see the checksum, do the following:

When you have scrolled to a specific program label (as in the above), press  **SHOW** (and hold the SHOW key down).

You will see, for example:

LBL A

LN = 24

CK = 9C33 Checksum in hexadecimal.


Press C to 'escape' from the list.

Note: A change in a variable name (say you had called something B but decide S would be better) does not change the length (i.e., the LN =) of a program. However, it does change the checksum.

Note: Length of the Program (LN) and the Checksum (CK) are covered on page B-2 of the User's Guide.

7.2 Deleting Programs from Memory

 MEM 2

Using the up and down cursor arrows ^ v select the label of the program you wish to delete. Then pressing  CLEAR will delete the program.

For example, to delete program A:

LBL A

LN = 24

 CLEAR

8.0 RCL(recall) and VIEW

Recalling a variable puts that value into the X stack register, while simply viewing a variable does not. One may, however, press ENTER after viewing a variable, which will then put the viewed value into the X stack register. (Although, this ENTER following a VIEW does not work within a program.)

Doing a recall of a variable into the X stack register, does not duplicate the value in the Y stack register. (It is not like typing a value and pressing ENTER.)

Two recalls in a row will fill the X and Y stack registers. The first recall ends up in Y stack register; the second recall ends up in X stack register.

Recalling a variable does not clear the recalled register. The recalled value is still there.

As an Aside: When doing a recall, the previous contents of X stack register is pushed up to the Y stack register unless the X stack register previously contained the 'default' zero, in which case the zero is over written. A 'default' zero is the result of a clearing operation. A zero which you have typed yourself will not be replaced/overwritten, but will move up to the Y stack register.


Note: There is a bit more to know about recall capabilities. This will be addressed below in Section 13.0.

9.0 Editing Programs and More Details about the Commands

This section will be better understood if you have at least two programs saved. So, let's enter two short programs.





First Program:

GTO .. GTO followed by two periods.


Note: The " GTO. ." to position the pointer can be done before or after the  PRGM command.

 PRGM

Takes you into the programming mode. (You will see PRGM TOP)

- A001  LBL A Gives the program a name (label), in this case 'A'.
- A002  INPUT D Permits you to enter a value for D. (You will be prompted to enter the value when the program runs.)
- A003  VIEW D Displays the value of variable D.
- A004  RTN Marks the end of the program.

Note: The RTN keeps this program from running into other programs.


Pressing C or pressing  PRGM again takes you out of programming mode.





Second Program:

Now, let's enter a second program. (It can, for simplicity, be the same program as the above just with a different label.) However, you should position the 'pointer' at the top of the memory before you enter the program.

Note: This second program is stored at the top. That is, above the first program in memory (higher up as opposed to below).

GTO .. Puts the pointer at the top.

 PRGM Takes you into the programming mode. (You will see PRGM TOP)

- B001  LBL B Gives the program a name (label), in this case 'B'.
- B002  INPUT E Permits you to enter a value for E. (You will be prompted to enter the value when the program runs.)
- B003  VIEW E Displays the value of variable E.
- B004  RTN Marks the end of the program.

Pressing C or pressing. PRGM again takes you out of programming mode.


9.1 Let's edit program B

Let's add another input statement and another view statement for variable F.

GTO . . will take you to the top of the program memory, or you can type GTO B ENTER. Although you won't be able to see the jump to program B (actually to line B001), when you eventually go into programming mode, you will be at B001.

Note: You can then use the v ^ cursor keys to scroll to the proper line or proper program.

9.2 Let's add the two lines


 PRGM You will see B001 LBL B in the lower window.

Use the down arrow 'v' till you see 'INPUT E' in the lower window.

B002 INPUT E

In program editing mode, typing a new command adds that command after the line appearing in the lower portion of the display.

Therefore type:


B003  INPUT F (The line number will automatically be inserted.)


Next scroll down till you see 'VIEW E'.

B004 VIEWE Was line B003, but will have automatically updated to B004 when the above line was inserted.

Therefore type:

B005  VIEW F

 PRGM or pressing C takes you out of programming mode.

Note: If you make a mistake, or want to delete a line, pressing <-- (above ) deletes the line in the lower portion of the display.

Let's run program B:

XEQ B ENTER

E? Enter any number and press R/S

F? Enter any number and press R/S

You will see E = With your value of E, then press R/S

You will see F = With your value of F, then press R/S

At this point our tiny program will end.

Note: After you enter a value, as opposed to simply pressing R/S, it is also possible to press ENTER followed by R/S. However, there is a subtle difference which will be covered later.

10.0 Before writing a more useful program, let's look at some commands

10.1 RTN (Return) Statement

You have already met the return statement. It marks the end of a program. Without the RTN, one program will simply run into the next program (but see note below). RTN also has another function which you will see in Section 11.0.

Note: A RTN statement is technically not absolutely necessary depending upon the nature of the last program statement. For example, you might have a GTO statement (see 10.2 below) which loops back to the start of the program. Just make sure that the current program will not run into the following program. Without the RTN, however, the program will not return to the beginning when it stops.

10.2 GTO (Go To) Statement

In addition to being an editing command to position the pointer in memory, GTO is also a programming command.

Note: In programming mode, you will notice a cursor (underscore) as you type the GTO. This is because it is expecting additional ‘information’ to follow. If you type only GTO A and press ENTER, it will assume A001, and the cursor will disappear.

Let’s say we want to edit program A so that it loops back to the input statement:

A001 LBL **A**

A002 INPUT **D**

A003 VIEW **D**

A004 GTO A002 Add this statement

A005 RTN


This program will then ask again and again for values of D, and then show you the values.

Note: The old values of D will still be there. Simply type a new value.

Simply press XEQ A ENTER to run — and C to escape from the program.

10.3 A Bit More About the GTO Statement

When in programming mode: A very clever feature of the HP 35s is that, if you change the program (add or subtract statements), the GTO statement automatically updates the line number of the GTO statement 'target'. (**Note:** I found that this works well, but it is not foolproof.)

When not in programming mode: That is, when you can see the X and Y stack registers, you can send the program pointer to a specific spot in your stored programs. For example, you can press GTO A002 ENTER. Then if you shift to programming mode  PRGM, you will be at line A002.

10.4 GTO (Followed and Not Followed by a Period)

In the HP 35s User's Guide, you may see the GTO command followed by a period and a line number: GTO. A002 for example. Let's look at the difference with, and without, the period, when in, and not in, programming mode.

10.4.1 When NOT in Programming Mode

The GTO. A002 (with the period) and GTO A002 (without the period) are the same when not in programming mode. The program pointer is positioned at A002 in both cases (although not visible, because you are not in programming mode).

10.4.2 When IN Programming Mode

The GTO. A002 (with the period) when in programming mode is an editing command, and the pointer jumps, right before your eyes, to that spot in the program memory; whereas, GTO A002 (without the period) simply becomes a statement in your program.

11.0 XEQ Statement (and an Additional Function of the RTN Statement)

You have already seen one application of XEQ, that is, in executing a program:

```
XEQ A ENTER
```

However, XEQ can also be a statement in a program.

Let's look at an example (again not very useful) but which illustrates the mechanics of XEQ's operation:

Note: You have seen the RTN statement in its role of ending a program (i.e., defining the end). Here, in association with the XEQ statement, the two statements function together somewhat like a subroutine. The RTN in this case 'remembers' where to return. Note: Below there are two programs, one starting with line numbers A and one starting with line numbers B.

```
A001 LBL A
```

```
A002 INPUT E
```

```
A003 VIEW E
```

```
A004 XEQ B    Will jump to program B at B001 (Line will display as  
                XEQ B001).
```

A005 VIEW **E**

A006 RTN

B001 LBL **B**

B002 INPUT **F**

B003 VIEW **F**

B004 RTN Returns to program A at A005 and then displays E
via the VIEW E command.

Note: You will need to press ENTER to complete the statement after typing XEQ B.

11.1 An Aside Regarding GTO and XEQ

The GTO, as mentioned above, and the XEQ when typed as program statements will have the cursor (underscore) trailing along. For example:

GTO A_ and XEQ A _


If ENTER is pressed, 001 will be filled in showing: GTO A001 and XEQ A001.


In other words: For any statement, when it is not obvious whether it is complete or not, it may be necessary to press ENTER. (This may also apply to entering numbers (constants) inside a program, but this will be discussed later.)


Note: While one usually sees, for example, XEQ A001, it is possible to execute a program starting from a different line number, for example: XEQ A005.


12.0 STO (Store) Statement


There are five forms of this statement. All are followed by a variable name (A to Z). **Note:** The first one below is without an operator before the variable name.

 **STO A** Copies the value in the X stack register to variable A.


 **STO + A** Adds the value in the X stack register to the value in variable A.

 **STO - A** Subtracts the value in the X stack register from the value in variable A.

 **STO ÷ A** Divides the value in variable A by the value in the X stack register.

 **STO × A** Multiplies the value in variable A by the value in the X stack register.

Note: In each case the result of the operation ends up being stored in the chosen variable (A to Z).

As an aside: In addition to the above variables A to Z, the functions (I) and (J) discussed in Section 20 can also be used as ‘variables’. For example:  **STO + (I)**

13.0 A Bit More About the RCL (Recall) Statement

There are five forms of the RCL statement. All are followed by a variable name (A to Z). **Note:** The first one below is without an operator before the variable name.


- RCL **A** Copies the A variable to the X stack register.
- RCL + **A** Adds the A variable to the value in the X stack register.
- RCL – **A** Subtracts the A variable from the value in the X stack register
- RCL ÷ **A** Divides the X stack register by the value in the A variable.
- RCL × **A** Multiplies the X stack register by the value in the A variable.

Note: In each case the result of the operation ends up in the X stack register. (The value in the chosen variable remains the same.)

We now have the commands necessary to do some simple straightforward programs. What we are lacking, at this point, are commands to modify the program during its operation. Let's take a look at a few commands with 'conditional' capabilities.

14.0 X?0 (Comparing the X Stack Register with Zero)

— The X?0 Menu

When you press  X?0 you will see a menu:

1 = \neq 2 = \leq 3 = $<$

4 = $>$ 5 = \geq 6 = $=$

To select one of the options, either enter the corresponding number, or use the arrow cursors to underline your choice and press ENTER.

We can use this feature to do some conditional branching.

Note: The X?0 decision rule is: Do [next command] if true. Otherwise skip the next command.

Note: See also the HP 35s User's Guide page 14.6

14.1 X?0 Example

Let's say you enter two numbers (D and E).

You want to take the difference (D – E) and store in F.

If the difference (D – E) is zero or negative you want to: look at the difference; change the sign; replace in F; see the number again; and start over.

For Example: Say that D=2 and E=5. You will see –3 change to +3.

If the difference is positive, you want to store in G; view G; add 10; store in G; view G; and start over.

For Example: Say the D=6 and E=1. You will see 5, then 15.


 PRGM

B001  LBL B Names the program



B002 INPUT D

B003  INPUT E

B004 – Does subtraction $D - E$.

B005  STO F Stores difference in F

B006 RCL F Recalls F

B007  $X \leq 0?$ To obtain this, press  $X?0$ followed by option 2. The ‘memory tool’ for $X?0$ is: “Do [next] if true.”

B008 GTOB010 Will go here if $X \leq 0$ is true.

B009 GTOB015 Will go here if $X \leq 0$ is false.

B010  VIEW F

B011 +/- Change the sign

B012  STO G

B013  VIEW G

B014 GTO B002

B015  STO G

B016  VIEW G

B017 RCL G

B018 10


B019 +

B020  STO G

B021  VIEW G

B022 GTO B002

B023  RTN

Note: If you do  MEM 2 and scroll to program B, you should see the following.

LBL B

LN = 71

CK = 7533 When you press  SHOW and hold down the .

Note: Strictly speaking, there are a few non-essential statements in the program. However, for now, I hope they make it easier to follow what is going on. We will address efficiency more as we progress.

Note: Regarding lines B008 and B009: It is common practice to follow the X?0 (as well as other HP 35s conditional commands) with two GTO statements.

14.2 Let's Run the X?0 Program

XEQ B ENTER

D? Type 2 then R/S to continue.

E? Type 5 then R/S

You will see: $F = -3$ (Because $2 - 5$ is negative.) R/S

Then you then will see: $F = 3$ (Now a positive number.) R/S

The program will repeat: (This time let's make the difference a positive number.)

D? 10 R/S (The 2 you typed earlier will still be there.
Just type a 10)

E? 2 R/S (The 5 you typed earlier will still be there.
Just type a 2)

You will see: $G = 8$ (Because $10 - 2$ is positive.) R/S

You will then see: $G = 18$ (Because 10 has been added to 8.) R/S

The program will then cycle back to ask for D.

Press C to escape from the program.

15.0 Looping Capabilities

With looping capabilities, we can write more complex programs.

15.1 ISG (Increment and Skip if Greater Than)

The ISG command functions as an automatic counter. It is commonly used in conjunction with two GTO statements.

The 'memory tool' is: Increment [the index variable] and Skip [the next command] if [the index variable is] Greater than [the specified limit].

The **↩ ISG** command requires a variable, say D, which contains the counter/index instructions.


For Example: **↩ ISG D**

Let's say we want to loop 5 times. We would store 0.005 in variable D.

The part after the decimal point (the number of times to loop) must be a three digit number. The value to the left of the decimal point is the starting number (which increments before starting).

15.2 Small Example to Shed Light on the **↩ ISG** Command

A001	↪ LBL A	Program A
A002	0.005	Information to loop 5 times
A003	↪ STO D	Stores 0.005 in variable D
A004	↩ ISG D	D tells ISG what to do
A005	GTOA007	Will be executed while counter (part before decimal point) is less than or equal to 5.
A006	GTOA009	Will be executed when counter value is greater than 5.
A007	↩ VIEW D	Should see counter value increment from 1.005 to 5.005
A008	GTO A004	Returns to ↩ ISG command.

A009  VIEW D Will be the sixth 'escape' value shown, should be 6.005

A010  RTN Return command.

Note: Be aware that the counter increments when line A004 is executed. That is, it increments before being evaluated.

Note Program Info:

LN = 35

CK = 02AD

15.3 What You Should See When Executing the ISG Program

1.005

2.005

3.005

4.005

5.005

The above represent the counter values while in the loop.

6.005

The above represents the counter value after escaping the loop.

The program then 'returns' and ends.

15.4 Let's Run the ISG Program

XEQ A ENTER

You will see:

D = 1.005 First time through loop (first digit has already incremented). R/S

D = 2.005 Second pass. R/S

D = 3.005 Third pass. R/S

D = 4.005 Fourth pass R/S


D = 5.005 Fifth pass R/S

Note: The counter is now greater than 5, and the program escapes the loop.

D = 6.005 The value outside the loop. R/S

The program now ends and the X and Y stack registers are displayed.

Note: Just as an aside, the values in the X and Y stack registers when the program finishes show 0.005. This is the value entered at the beginning of the program. The view commands do not enter values into the X stack register.

Note: A common programming approach with  ISG is to follow with two GTO commands.

Note: For more ISG details see page 14.18 of the HP 35s User's Guide. There is also a good looping example in the HP 35s User's Guide under Conditional Loops (GTO) on page 14.17.

16.0 Guessing Game

Let's now combine the last two commands (X?0 and ISG) to make a simple guessing game. The first 'player' will enter a single digit whole number 'N' between 0 and 9. The second 'player' will have four chances 'G' to guess the number. If they guess correctly the display will show 999. Note: Shortly, rather than displaying 999, you will see how to display a message like 'WINNER'.


B001 LBL B

B002 0.004 Control for the ISG statement.

B003  STO D Stores control in D for 4 guesses.

B004 999 Displayed value for a correct guess indicator.







B005  STO F Stores 999 value to display.

B006  INPUT N Opponent enters number 'N' for you to guess.

B007  ISG D Starts looping counter operation.

B008 GTOB010 Goes to B010 if $D \leq 4$

B009 GTO B020 Goes to B020 if > 4 . Remember: ISG "skips next if > 4 ".

B010	 INPUT	G	Enter your guess (you get 4 tries).
B011	RCL	N	Will recall your opponent's chosen number.
B012	-		Y stack register minus X stack register.
B013	 STO	E	Stores difference in E.
B014	 X=0?		X?0 Option 6 Checks if E is zero. "Do next if true."
B015	GTOB017		Do this if E = 0.
B016	GTOB019		Do this if E ≠ 0
B017	 VIEW	F	Will display 999 (you guessed correctly).
B018	 RTN		Program will end.
B019	GTOB 07		Goes back to ISG, and you guess again.
B020	 RTN		You used all 4 guesses. Program ends.

LN = 92

CK = 139C


Note: Our program is not very sophisticated. It doesn't check for the correct range of integer numbers (in this case 1 to 10). The program doesn't have word prompts, so you have to remember what to enter just by the variable names N and G. We will address these shortcomings as we proceed.

This program does require some knowledge of the stack operation. For example lines B010 to B013. At B010 you enter your guess. It goes into G, but also into the X stack register. The RCL of N at B011

goes into the X stack register pushing the previous value to the Y stack register. The subtraction at B012 puts the difference (i.e., Y-X) into the X stack register, which is saved into E at B013 (which strictly speaking isn't necessary). One other thing: at B014 the X stack register is actually being evaluated, as opposed to E, if one is being technical.

Note: Later there will be a single player modification for the above program, where the number to guess will be chosen at random. See Section 21.0

17.0 X?Y (Comparing the X Stack Register With the Y Stack Register)

When you press  X?Y you will see a menu:

1 = ≠ 2 = ≤ 3 = <

4 = > 5 = ≥ 6 = =

To select one of the options, either enter the corresponding number, or use the arrow cursors to underline your choice and press ENTER.

We can use this feature to do some conditional branching.

Note: The X?Y decision rule is: Do [next command] if true. Otherwise skip the next command.

Note: See also the HP 35s User's Guide pages 14.7, 14.8

With our new **↩ X?Y** command, we have increased our programming flexibility. We can now directly compare the X and Y stack registers, whereas in the Guessing Game (Section 16.0), we had to first compute a difference, and then compare that difference with zero using X?0.

17.1 Updated Guessing Game with **↩ X?Y**

The new Guessing Game commands start at B009.

B001 LBL **B**

B002 0.004 Control for the ISG statement.

B003 **↪ STO D** Stores control in D for 4 guesses.

B004 999 Displayed value for a correct guess indicator.

B005 **↪ STO D** Stores 999 value to display.

B006 **↩ INPUT N** Opponent enters number 'N' for you to guess.





B007 **↩ ISG N** Starts looping counter operation.

B008 GTOB010 Goes to B010 if $D \leq 4$

B009 GTO B018 Goes to B018 if > 4 . Remember: IG "skips next if > 4 ".

B010 **↩ INPUT G** Will end up in Y register.

B011 RCL **N** Will be in X register

B012	 X?Y	X?Y with option 6. (Decision rule: “Do next if true.”)
B013	GT0B015	Do this if X = Y.
B014	GT0B017	Do this if X + Y.
B015	 VIEW F	Will display 999 (you guessed correctly.
B016	 RTN	Program will end.
B017	GTO B007	Goes back to ISG, and you guess again.
B018	 RTN	You used all 4 guesses. Program ends.

LN = 62

CK = A979

Note: The entry order of X and Y makes a difference (but not in the present case). For example, with option 3, (the < option), the evaluation will be true if the X stack register is less than the Y stack register.

18.0 Simple Program with a Few Commands Not Yet Used (Including Storage Arithmetic)

Below is a simple program to demonstrate a few commands not yet used. (Again it is not a very useful program, mainly because there is a built-in statistical routine that does the same thing: (HP 35s User’s Guide, page 12.1)




This program allows you to enter a series of numbers, one at a time, and when you finish you can view the sum, sum of squares, and N (the number of observations entered).


This is the first program which contains a number in among the commands (see lines A004, and A007).

Make sure registers A through C are set to zero at the start. (Note: This can be done automatically which will be shown below under Clearing Variables.)


The program stores the sum in variable A, the squares in variable B, the number of observations in variable C.

Note: Don't forget the plus signs in lines 3, 6, and 8.

A001	 LBL C	Names (labels) the program.
A002	 INPUT D	Enter your observation
A003	 STO + A	Add X stack register to variable A (sum of observations)
A004	2	Displayed in X stack register (Sets up squaring operation)
A005	y^x	Squares value in Y stack register (Value entered at A002)

A006  + B Sums values in X register (sums of squared observations)

A007 1 Value for counter (increment value)


A008  + C Stores number of observations (times through loop)

A009 GTO A002


A010  RTN

Specifications to check entry of program: LN = 32, CK = 8969

Note: You can now (in non-programming mode) VIEW the variables: A (sum), B (sum of squares), and C (n, number of observations).

Clearing Variables:  CLEAR 2 Option 2 (This will display in a program as 'CLVAR\$'.) Note: This clears only the direct variables A through Z. You could make this the second line (before INPUT) in the program above.

19.0 More Looping capabilities: The DSE (Decrement, Skip [next command] if Equal or Less Than)

The following presents the minimum information required to use the  DSE command. For more details, see page 14.18 of the HP 35s User's Guide.

The 'memory tool' is: Decrement [the index] and skip [the next command] if less than or equal to [the reference value].


Below is a small example to shed light on the **↷ DSE** command. Let's say we want to loop five times. The **↷ DSE** command is followed by a variable which contains the counter instructions.


We store 6.000 in variable B, which follows the command, **↷ DSE B**. (Note: To loop five times we indicate 6.000 and not 5.000)

Note: The part before the decimal point indicates the starting value of the counter and will decrement before starting. The part following the decimal point (the reference value) must be a three digit number.

19.1 DSE Program to Loop Five Times

B001	↷ LBL B	Program B
B002	6.000	Value to loop 5 times (Note the value of 6 and not 5.)
B003	↷ STO D	Stores 6.000 in variable D
B004	↷ DSE D	
B005	GTOB007	Will be executed while counter value is greater than 000.
B006	GTOB009	Will be executed when counter value is equal or less than 000.
B007	↶ VIEW D	Should see counter value decrement 5.000 to 1.000
B008	GTOB004	Returns toll DSE command.

B009  VIEW D Will be sixth value shown, should be 0.000

B010  RTN Return command.

LN = 35 CK = BDD8

Note: Be aware that the counter decrements when B004 is executed. That is, it decrements before being evaluated.

19.2 Let's Run the DSE Program

XEQ B ENTER

Note: Press the R/S key after each display. The first five are 'views' from line B007:

D = 5.000 First time through loop (first digit has already decremented).

D = 4.000 Second pass.

D = 3.000 Third pass.

D = 2.000 Fourth pass

D = 1.000 Fifth pass.

Note: The counter is now equal or less than the value 000, and the program escapes the loop.

D = 0.000 View from line B009 outside the loop.

The program now ends and the X and Y stack registers are displayed.

Note: Just as an aside, the value in the X stack register when the program finishes shows 6. This is the value entered at line B002.

Note: Again, as with the ISG command, a common programming approach when using the DSE command is to follow it with two GTO commands.

Note: For more complete information on DSE see page 14.18 of the HP 35s User's Guide.

20.0 Addressing Direct and Indirect Variables - The Functions (I) and (J)

20.1 Use of variables I and J and the functions (I) and (J)

The direct variables I and J can be used with the function keys (I) and (J) to address other variables, both direct and indirect.

For example: If $J = -1$, then the function (J) represents variable A. If $J = -2$ then the function (J) represents variable B, etc. Note the negative signs.

Note: -1 addresses variable A, -2 addresses B. .. -26 addresses Z.



20.2 Example Referencing Direct Variables

-1 Note the minus sign. Use +/- key to obtain the minus.

 STO I Letter I

123

 **STO (I)**

Note: You don't have to precede (I) with the  or  key.

Essentially the above says to store 123 in variable A, since I = -1 references A.

-2

 **STO J**

Letter J

456

 **STO (J)**

Essentially the above says to store 456 in variable B, since J = -2 references B.

 **VIEW A** or  **VIEW (I)** (Will display 123)

 **VIEW B** or  **VIEW (J)** (Will display 456)

Note: Using the above technique you can access an additional 801 indirect variables (discussed below).

20.3 Example Addressing Indirect Variables


When I or J is set to 0 through 800, the (I) and (J) functions reference the corresponding indirect variables 0 through 800. (Recall that -1 through -26. i.e., with minus signs, reference the direct variables A through Z)

Note: I don't suggest using the indirect variable 0, in that the clearing of that variable requires an extra step. See Section 20.8.

1

 Direct variable I = 1


123

 Stores 123 in the indirect variable 1 via (I) function.

2

 Direct variable I = 2


456

 Stores 456 in the indirect variable 2 via (I) function.

20

 Direct variable I = 20

987

 Stores 987 in the indirect variable 20 via (I) function.

Note: You can also use (I) and (J) like regular variable names. For example:

VIEW (I), STO (I), RCL (J), STO + (J), INPUT (I), etcetera.

Note: One might think, as a first impression, that there are 2 X 801 indirect variables, one set associated with (I) and a second set associated with (J). This is not the case. For example, if I and J are both set to 1, and you store, say, 999 in (I), if you then view (J) you will see the same 999.

20.3.1 Indirect Variable Storage Requirements






Each indirect variable requires 37 bytes of storage. Contrast this with a single programming line which requires only 3 to 4 bytes, or

numbers in programs which require a varying number of bytes depending upon the number of digits. For example: 1 digit (0 – 9) = 4 bytes; 2 digit (10 – 99) = 5 bytes; 3 digit (100 – 999) = 6 bytes, etcetera.

As an Aside: If, for some reason, you store a value in, let's say, the indirect variable location (500), upon doing this, memory is immediately also allocated for all indirect variables below that location (i.e., 0 to 499). Therefore, if you assigned location 800, i.e. the maximum limit, you will allocate essentially the total amount of memory available in the calculator. That is: $801 \times 37 \text{ bytes} = 29,637 \text{ bytes}$. Recall the total available is 30,192.

20.4 Small Program to Input Data into Indirect Variables

Note: Let's enter: 2, 4, 6, 8, 10 when asked.

- E001  LBL E Program E
- E002 1 Increment value for indirect variables.
- E003  STO + I Storage Addition adds increment value to variable I. (See Note below.)
- E004  INPUT N Asks for data value.
- E005  STO (I) Stores data value as an indirect variable.
- E006 GTOE002 Loops back to start process again.
- E007  RTN Return marks end of program.








Program specifics: LN = 22 CK = DB6A



Note: This is a barebones program which assumes certain things have already been done or specified — things which could be automated. For example, it assumes variable I is cleared (i.e., starts at zero). You could start at another location if desired, for example, to use higher registers.

20.5 Small Program to Read Data from Indirect Variables

(Program sums the recalled data into variable A; the sums of squares into variable B; the count of the number of observations into variable C.)

Note: Make sure variables A, B, C, and I are cleared before starting.

F001	 LBL F	Program F
F002	1	Increment (index) value for indirect variables (I) function.
F003	 STO + I	‘Storage Addition’ which adds increment value to variable I.
F004	 VIEW I	Basically, to see where you are.
F005	RCL (I)	Recalls the data value from storage.
F006	 VIEW (I)	Basically to see the recalled data value.
F007	 STO + A	To store sums.
F008	 X²	To square values.
F009	 STO + B	To store squares.

- F010 1 Value for following command to count.
- F011  STO + C To store number of observations.
- F012 GTOF002 To loop back to beginning.
- F013  RTN Marks end of program.

Program specifies: LN = 41 CK = FF8A

Note: The above is still a rather primitive program. That is, you must make sure certain registers are cleared before you start. Also, you must know when to stop processing the data, that is, the number of data values which were stored. And you must view the results stored in variables A, B, and C when the program finishes. These are all factors which could be automated.

20.6 Let's Run the Above Program to Read Data from the Indirect Variables

Note: First clear VARS:  CLEAR 2 Option 2

Remember the data entered were 2, 4, 6, 8, and 10.

XEQ F ENTER

You will see:

I = 1. Press R/S from VIEW I, the observation number.

(I) = 2. Press R/S from VIEW (I), the actual observation.

I = 2. Press R/S

(2) = 4. Press R/S

I = 3. Press R/S

(3) = 6. Press R/S


I = 4. Press R/S


(4) = 8. Press R/S


I = 5. Press R/S

(5) = 10. Press R/S

I = 6. **At this point press C (to exit the program).**


Now  **VIEW** A. Should be 30 Sum of the observations.

Now  **VIEW** B. Should be 220 Sum of the squared values.

Now  **VIEW** C. Should be 5 Number of observations.

20.7 To Erase/Clear Indirect Registers

To clear the indirect registers, do the following:

 **CLEAR** 6 then 000 (Enter option choice 6, then after CLVAR, enter 000)

This clears all registers above 000, (i.e., 1 to 800), but not register 0. This is the reason I do not suggest using register 0.

See: Keys for Clearing (continued): HP 35s User's Guide, page 1.5.

20.8 Clearing the 0 Indirect Register

When you check MEM after clearing as in Section 20.4, it will still show 1 indirect variable being used (i.e., register 0). To clear the register, do the following:

0 Zero


 **STO I** Direct variable I = 0

0 Zero

 **STO (I)** Stores 0 in indirect register 0 via (I) function.

To see if it is cleared:

 **VIEW (I)** If it says, “INVALID (I)” it is has been cleared.

Note: Repeating something mentioned earlier, the number of indirect variables used actually references the index number of the highest variable used. For example, if you store a number in the indirect variable 20 (and don't even use 0 to 19), when you check the memory  **MEM**, it will show 21. That is, it indicates that 0 to 20 are in use.

21.0 Addition to the Guessing Game (Section 16.0) using RAND and INTG

In Section 16.0 we had a guessing game where your opponent enters a single digit whole number N between 0 and 9 for you to guess (see line B006). We can modify this program such that the number to guess is entered at random by the program. You would insert the following between lines B005 and B007 to replace line B006. (Of course, this will affect the line numbers as well as the GTO statements which will have to be modified to reflect this.)

The following are the statements to insert, but see the explanation below of what takes place in the stack.

 **RAND**

Appears as RANDOM in the program.

10

X


 **INTG** 6

Option 6 to obtain the Integer Part of 5.850. (It will appear as 'IP' in the program.)

 **STO** N

Stores the number chosen at random for you to guess.

Activities in the stack:

 **RAND** gives a number > 0 and < 1 , for example, 0.5850, which will appear in the X stack register. (For RAND, see HP 35s User's Guide page 4.15 and page B-4)

0.5850 From RAND


5.850 Result of multiplying 0.5850 by 10


5.0000 Integer portion, the result of  **INTG** 6

 **STO** N

Stores the number chosen at random for you to guess.

Note: For more on INTG, see HP 35s User's Guide pages 1.6, 4.2, and C-4.

Note: RAND works in combination with  **SEED**. If you start with a specific seed number, the same sequence of pseudorandom numbers will be generated.

To set the seed, enter a number, for example 976, and press  **SEED**. See page 4.15 of the HP 35s User's Guide for details.

22.0 Equation Solver (When the Equations Are Stored In the List)

See also the HP 35s User's Guide Sections 6 and 7.

22.1 Entering an Equation

To start the process, **press EQN** to trigger the equation input mode and to see the list of the built in equations.

As you use the "arrow" keys to scroll, you will see:

EQN LIST TOP

2*2 lin solve (a built in)

3*3 lin solve (a built in)

With the 3*3 lin solve in the lower window, let's type in our formula.

A new line will automatically be started when you begin to type.

Note: Equations are typed as if you were in algebraic mode.

Let's say we want to type: $A = B + C$

Note: To type the variable names (i.e. letters A to Z), precede the letter with the RCL key.

RCL **A**  = RCL **B** + RCL **C** (**Note:** You will only see $A = B + C$)

Notes:

Blinking Cursor: After the above **C** is typed, the cursor ' _ ' will still be blinking. Press ENTER to complete the formula.

Editing: You can use the cursor arrows and the <-- (arrow above the **CLEAR**) key to edit the EQN lines.

Deleting Equations from the List: Scroll the equation you would like to delete into the bottom line of the display, then press the <-- (arrow above the **CLEAR**) key.

Hidden Symbol Key: The Y^X key inserts the sign. (As in 3 squared: 3^2 .)

Exiting: Press C to exit the equation mode.

22.2 Three Approaches When Evaluating Equations

22.2.1 Method 1. EQN followed by ENTER

See also the HP 35s User's Guide page 6.3

With the first approach, the **EQN command is followed by ENTER.**

As a simple introduction, say you want to evaluate A in the following equation:

$A = B + C$ (Entered per Section 22.1 above)

To execute the program:

Press EQN to see the list of equations if they are not already visible. Then using the cursor keys, scroll until the desired equation ($A = B + C$) is in the bottom window.

ENTER Press ENTER to evaluate the equation.

You will be prompted for those variables on the right side of the equation:

B? For example type 2 and press R/S (i.e., Run/Stop)

C? Type 3 and press R/S

You will then see $A = 5$

Press C to leave equation mode.

Note: The 5 (from $A = 5$) will also be in the X stack register.

Note: The A, B, and C are direct variables and will retain their values after you exit the EQN mode.

Editing the Equation List: You can use the cursor arrows and the <-- (arrow above the CLEAR) key to edit the EQN lines.

Deleting equations from the list: Scroll the equation you would like to delete into the bottom line of the display. Then press the <-- (arrow above the CLEAR) key.

Press C to leave equation mode.

22.2.2 Method 2. EQN followed by SOLVE

With the second approach the **EQN command is followed by SOLVE**.

You enter the equation as demonstrated in Section 22.1 above.

Let's assume the equation is $A = L \times W$

Using the SOLVE approach, the program will ask which variable you would like to solve for.

To execute the program:

Press EQN to go to the list of saved equations if they are not already visible.

Scroll to see $A = L \times W$ in the lower window.

Press  **SOLVE** and you will see:

SOLVE _ Let's say you type W.

A? Say you type 100 and press R/S.

L? Say you type 20 and press R/S. You will then see:

W = 5 That is, when $A = 100$ and $L = 20$, then $W = 5$.

Press C to leave the equation mode.

Note: The 5 (from $W = 5$) will also be in the X stack register.

22.2.3 Method 3. EQN followed by XEQ

With the third approach the **EQN command is followed by XEQ.**

This approach is a bit different from those shown above. This approach wants to either:

1.) Evaluate an expression (for example: $A + B + C$),

or

2.) If the expression includes an equal sign, for example: $A = B \times C$, this approach wants to balance both sides of the equation by giving the adjustment for the right hand side.

You enter the equation as in Section 22.1 above.

Let's take the first case ($A + B + C$): If $A = 1$, $B = 2$, and $C = 3$, you will see 6 in the X stack register.

Let's take the second case ($A = B \times C$): If $A = 1$, $B = 2$, and $C = 3$, you will see -5 (i.e., minus 5) in the X stack register. In other words if the expression is $1 = 2 \times 3$, then -5 is the adjustment to make the right side equal the left side. That is $1 = (2 \times 3) - 5$.

To execute the program:

Press EQN to go to the list of saved equations if they are not already visible.

Now, let's take the first example: $A + B + C$. (Note: No equal sign.)

Let's say $A = 1$, $B = 2$, and $C = 3$.

Scroll to see $A + B + C$ in the lower window.

Press XEQ and you will see:

A? Type 1 and press R/S.

B? Type 2 and press R/S.

C? Type 3 and press R/S. You will then see in the X stack register:

6 That is: $1 + 2 + 3$

Press C to leave the equation mode.

Now, let's take the second example: $A = B \times C$

Scroll to see $A = B \times C$ in the lower window.

Press XEQ and you will see:

A? Type a 1 and press R/S.

B? Type 2 and press R/S.

C? Type 3 and press R/S. You will then see in the X stack register:

-5 That is, subtract 5 from the right side to make things equal:

$1 = (2 \times 3) - 5$

Press C to leave the equation mode.

Note: All of the above methods put the answer into the X stack register, but only the first and second display the answer during execution.

Note: Section 30.2 discusses an example from the HP 35s User's Guide which is a bit more complicated.

23.0 Equation Solver (When the Equations Are NOT Stored in the Equation List)

23.1 FN = Program Label

You have already seen a bit about how to use the equation solver when the equations are stored in the Equation List. Now let's take a look at solving equations which are embedded in a program.


Take the equation $A = B + C$ which is rather trivial, but its difficulty is not the focus right now.

First we have the program that contains the equation:

 **LBL K**

 **INPUT B**


 **INPUT C**


$A = B + C$ Press EQN, then RCL **A**  = RCL **B** + RCL **C** then ENTER to complete.

 **RTN**

Now the program that calls the above:

 **LBL L**

 **FN K** FN refers to a programmed function. K is the above program name.

 SOLVE A The variable to solve. (Also see ** below.)

 VIEW A

 RTN

Let's run our program:

XEQ L ENTER L is the 'calling' program.

B? Program asks for value of B. Type a 5 followed by R/S

C? Program asks for value of C. Type a 2 followed by R/S

You will see 'SOLVING' followed by:

A= 7 Press R/S to finish.

Note: I recommend that you now skip directly to Section 23.2, at least if this is your first reading of the book.

Although flags are not discussed until Section 24.0, I wanted to make sure that a few comments were made here regarding their connection with equations. In short, make sure that you don't have Flag 10 set, which would cause the program to ignore the processing of the equations.

Also: One could eliminate both input statements in program K if Flag 11 is set. With flag 11 set, the program will prompt for both B and C. However, since flag 11 automatically clears itself after a program runs, it is useful to insert the second line below to make sure that the flag is set, (if that is desired):

 LBL K First line

↵ **FLAGS** followed by option 1, followed by . 1 (i.e., a period followed by a one). It will appear as SF11 (Set Flag 11) in the program.

A = B + C Press EQN, then RCL A **↵** = RCL B + RCL C then ENTER to complete.

↵ **RTN**

** The SOLVE can be replaced by ENTER (and no variable name), in which case the command acts in a manner similar to XEQ.

23.2 How the 'FN =' Works Outside the Programming Mode

(Note: The following assumes you have program K)

↵ **FN** **K** From the keyboard (i.e., outside programming mode).

↵ **SOLVE** **A** From the keyboard (i.e., outside programming mode).

B? You will be prompted (since flag 11 is set). Type 5 then R/S

C? You will be prompted. Type 10 then R/S

A = 15 Answer

Of course you could also solve for B or C.


As an Aside: This 'FN =' approach has some advantages, under certain conditions, as opposed to using the Equation List. For example: Let's say you have, once again our trivial equation $A = B +$

C. However, if B is affected by D, E, and F, only two of which you need depending upon other factors, these computations and conditional decisions can be made in program K while leaving program L (or the 23.2 approach above) quite simple in form.

24.0 FLAGS

The HP 35s has 12 flags numbered 0 through 11. They are either 'set' or 'cleared'. Before any specifics, let's look at the flag menu.

24.1 Flags Menu

To see the menu, press  **FLAGS**

You will see:

1 SF 2 CF

3 FS?

The first option SF stands for Set Flag. Pressing 1 results in seeing SF __. You can then enter a number between 0 and 11. (Note: For numbers 10 and 11, you type a period for the 1 in the tens position, followed by either a 0 or a 1. That is: period 0 = 10, and period 1 = 11.)

The second option CF stands for Clear Flag. This operates in the same fashion as the above, but clears the indicated flag.


The third option FS? stands for the question: [Is the] Flag Set?. This queries the flag in which you are interested. The response when not in program mode is displayed as YES or NO.

Note: In programing mode the response to FS? is either to: do the next command if the flag is set, or skip the next command if the flag is not set (i.e., it is cleared). The memory rule is: Do next [program command] if true [that the flag is set]. If the flag is not set, the immediate following command is skipped. Keep in mind that flags can be set or cleared from within a program, or externally to a program.

24.2 FLAGS 0 through 4

Flags 0 through 4 have no preassigned meanings. They are available like 'switches' in a program that can be queried and operated upon. You might have a rather complex program capable of doing a number of things. Using the flags, you can, for example, turn on or off certain paths through the program to select only the desired outputs. They might also be used during debugging. A flag might be set or cleared if the program execution takes an unusual turn of events.

A typical statement, for example, to set flag 4 in a program would be entered as follows:

-  **FLAGS 1** Option 1 to set a flag.
- SF __ An SF with an underscore appears in the program.
Type a 4 to set flag 4.
- SF 4 This is how the statement will appear in your program.

24.3 FLAGS 5 through 9

Flags 5 through 9 have to do with error and overflow messages and the display of fractions. These will not be discussed. See the HP 35s User's Guide page 14.9

24.4 FLAGS: 10 and 11

Flags 10 and 11 have to do with the equation solver (EQN) discussed in Sections 22.0 and 23.0. These two flags are a bit confusing, and will therefore be our focus.

When flag 10 is set, equations are not processed, but simply displayed like text. The situation is usually such, that the 'equation' is some message or prompt which it is desired to display.

When flag 11 is set, the program prompts for the variables involved in the equation.

Note: Flag 11 is automatically cleared after the program (i.e., the equation) executes. Therefore if the same program is executed more than once, the flag is cleared unless reset. (Discussed near the end of Section 23.1 under 'Also.')

Let's say we have the simple equation:

$A = B + C$ (Where $A = 0$, $B = 10$, and $C = 5$, just to have some specific values.)

We will consider six situations when using the equation solver:

With and without flag 10 being set:

With and without flag 11 being set:

With and without a SOLVE statement in the program:

24.4.1 With Flag 10 set; Flag 11 set; With 'SOLVE = A' statement in the program

Formula will display; program will ask for B and C; formula will display again; you will see SOLVING — and program may 'hang up'. You can press C to interrupt. **Note:** If variables A, B, and C don't have any values (other than zero), program simply stops and shows 0 in the X stack register.

24.4.2 With Flag 10 set; Flag 11 set; Without the 'SOLVE = A' statement in the program

Formula will display; X register is displayed; formula displays; X register is displayed;

Conclusions with Flag 10 set: With flag 10 set, the formula displays, but EQN doesn't process.

24.4.3 With Flag 10 clear; Flag 11 clear; With 'SOLVE = A' statement in the program

Program will ask for B and C; you will then see SOLVING; then you will see 15 in X register;

24.4.4 With Flag 10 clear; Flag 11 clear; Without 'SOLVE = A' statement in the program

Displays RUNNING briefly, then displays the X register; **Note:** If variable B and C are preloaded with, say 10 and 5, the answer appears in X register as -15. (This assumes variable A = 0. This is explained below.) **Note:** With neither flag 11 set, nor a SOLVE statement there was no prompting for variables.

24.4.5 With Flag 10 clear; Flag 11 set; With 'SOLVE = A' statement in the program

Shows RUNNING briefly; then asks for A (let's say 0); then asks for B (say 10); then C (say 5); asks for B again; asks for C again; then displays SOLVING; then 15 is displayed in X register; **Note:** Even if you give a value of 2 for A, the answer in the X stack register is still 15. **Note:** Flag 11 is cleared when program executes. **As an aside:** Flag 11 is responsible for the first round of prompts; SOLVE is responsible for the second round.

24.4.6 With Flag 10 clear; Flag 11 set; Without 'SOLVE = A' statement in the program

Program will asks for A (let's say 0); B (say 10); C (say 5); will show -15 in X register. **Note:** This time (without the SOLVE), if I enter A as 2, the answer of -13 appears in the X register (see reason below). **Note:** Flag 11 is cleared when program executes.

Conclusions with Flag 11 set: (Assuming A = 0, B = 10, C = 5)

Asks for all variables A, B, and C.

With SOLVE A: Gives 15 in X stack register.

Without SOLVE A: Gives -15 in X stack register.

Conclusions with Flag 11 clear:

With the SOLVE A statement: Program asks for only B and C; then shows 15 in X stack register.

Without the SOLVE A: Program uses any values previously stored in A, B, and C, and shows -15.

Regarding the Negative Values:

Without a SOLVE statement specific to a particular variable, for example 'SOLVE A', the EQN procedure does essentially the following:

Let's assume: $A = 2$, with $B = 10$, and $C = 5$ for the equation $A = B + C$, in other words $2 = 10 + 5$.

The 2 is moved to the right side with the sign changed: $10 + 5 - 2$

To make the right side (and therefore both sides) equal to zero, the solution is to add minus 13 to the right side. Therefore -13 would appear in the X stack register.

For more information on specific flags, see the User's Guide pages 14.9 through 14.16


25.0 Displaying Messages in Programs (FLAG 10 and EQN)




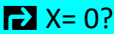



In Section 16.0 there was a guessing game. In that program if you guessed correctly a 999 was displayed. In this section we will use a FLAG 10 command and an equation (EQN) to display the word 'WINNER'. Setting FLAG 10 instructs the equation processor to not actually process the equation, but simply display it.

L001 LBL L

L002 0.004 Control for the ISG statement.

L003  STO D Stores control in D for 4 guesses.

L004  INPUT N Opponent enters number 'N' for you to guess.

L005	 ISG D	Starts looping counter operation.
L006	GTOL008	Goes to L008 if $D \leq 4$ (You get to guess.)
L007	GTO L019	Goes to L019 if > 4 . Remember: ISG “skips next if > 4 ”.
L008	 INPUT G	Enter your guess (you get 4 tries).
L009	RCL N	Will recall your opponent’s chosen number.
L010	–	Y stack register minus X stack register.
L011	 STO E	Stores difference in E
L012	 X= 0?	X?0 Option 6 Checks if E is zero. “Do next if true.”
L013	GTOL015	Do this if $E = 0$. (Number guessed correctly)
L014	GTOL018	Do this if $E \neq 0$
L015	SF10	 FLAGS, then option 1 to set flag, then . 0 (i.e., period zero).
L016	WINNER	Press EQN, then RCL before each letter, then ENTER to complete. Note the underscore: (WINNER_ ENTER to turn off the EQN mode.)
L017	 RTN	Program will end.
L018	GTOL005	Goes back to ISG, and you guess again.
L019	 RTN	You used all 4 guesses. Program ends.

Note: Program specifics: LN = 68, and CK = AAAA

See the User’s Guide Section 14.11 and 14.14 regarding Flag 10.

26.0 Equivalent Traditional Programming Commands

Although at first glance the HP 35s doesn't appear to have traditional style programming statements, it does have a number of equivalent statements.

26.1 Assignment Statements

Left is a version of the traditional assignment statement — Right is the HP 35s equivalent (with commas separating the commands).

$A = 15$	15, STO A
$A = A + 15$	RCL A, 15, STO + A
$A = A + B$	RCL A, RCL B, +, STO A
$A = 3 + 4$	3, ENTER, 4, +, STO A

26.2 COMMENTS

I haven't found anything for the HP 35s similar to the traditional program comment statement found in other programming languages. One can make short messages appear during program execution using EQN with the FLAG 10 set (See Section 25.0), but other than this, which is not really a comment statement, I haven't found anything.

26.3 ARRAYS and INDICIES

While there are no arrays and dimensions, as such, one can use the (I) (J) functions, along with the direct and indirect variable addressing, to simulate arrays, if one is clever.

26.4 IF STATEMENTS

The X?Y and X?0 operations are the closest thing to the traditional IF statement. With their various 'equality/inequality' options you have a great deal of flexibility. Keep in mind that the memory rule is: Do next statement if true. (If false, the next statement is skipped over.) See 28.0 for some IF, THEN, ELSE equivalents.

26.5 DO LOOP

There are no do loops, as such, but the ISG (Section 15.0) and the DSE (Section 19.0) commands perform a similar function. See also section 27.0 for an approximate HP 35s do loop approach.


Note: Contrast the above IF STATEMENT rule with the rules for the ISG (Increment and Skip if Greater than) and DSE (Decrement and Skip if Equal or less) commands. Here the next command is executed unless the index value exceeds the cutoff value.

26.6 Go To

In this case, the HP 35s has almost an exact equivalent with its GTO command.

Note: In most cases, when a program is edited, the GTO statements will automatically be updated with the correct ‘go to addresses’. However, I have updated some programs where the addresses have gotten mixed-up. I haven’t been able to pinpoint when exactly this occurs, but perhaps when I started editing at the bottom of the program, or jumped around, making the changes.

26.7 WRITE / PUT / PRINT

Probably the closest thing although only a display — would be the  **VIEW** command. It is also worth knowing that, when outside the programming mode, if you press ENTER after VIEW, you can ‘paste’ the viewed result into the X stack register. This does not work, however, in programming mode when an ENTER statement follows a VIEW statement. In this situation, any value currently in the X stack register is simply entered again.

27.0 HP 35s ‘Do Loop’ Approach Example For Example, DO A = 1 to 5 (Featuring the X?Y Command)


N001  **LBL** N


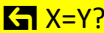


N002 0 To set up lower index.

N003  **STO** A To store the lower index.

N004 5 To set up upper index.

N005  **STO** B To store the upper index.

N006  **INPUT** S Just some operation to perform as a reference point.




N007 1	Increment value.
N008  STO + A	Add increment value to A.
N009 RCL A	Recall current increment value.
N010 RCL B	Recall upper index.
N011  X=Y?	X?Y with option 6 To test if increment value equals upper index. (Memory tool: Do next [command] if true.)
N012  RTN	Ends the program since upper index was reached.
N013 GTON006	Goes back to INPUT statement.
N014  RTN	Bottom of program marker.

28.0 IF THEN ELSE Equivalents (Using the X?0 options)

Note: Only the skeleton of a program is shown.

Assume a value has been entered into the X stack register (with any necessary computations performed) for a comparison with zero below.

{Miscellaneous previous statements here}

A005 X<0?	The  X?0 with option 3. Remember: Do next if true.
A006 GTOA012	Do this if above is true.
A007 X=0?	The  X?0 with option 6.
A008 GTOA013	Do this if above is true.
A009 X>0?	The  X?0 with option 4.

A010 GTOA014 Do this if above is true.

A011 GTOA015 Almost like an ELSE statement, but unnecessary here.

Note: Strictly speaking, not all of the above statements are needed. That is, if X is not less than or equal to zero, it must be greater than zero. Also, if no alternatives exist, an else is unnecessary.

29.0 Somewhat 'Mysterious' Commands

I wasn't sure whether to call this section Odds and Ends or something else. In any case, I've put a few items here which just seemed to clutter other sections.

29.1 Pause: PSE (pause)

In general, a PSE command in a program will cause the program, when it executes, to display the contents of the X stack register for about one second.

Also, if PSE follows a VIEW statement, that VIEW will only remain for about one second. Otherwise, a VIEW will temporarily stop the program execution and require that you press R/S to continue.

Note: For some exceptions and additional information, see the HP 35S User's Guide, pages: 13.19, 14.11, and 15.10. Page 13.19 is very informative.

29.2 RTN and STOP

There is no STOP key on the calculator. In programming mode the R/S key inserts the STOP command.

In general, both the RTN and the STOP command terminate a program's execution. However, RTN repositions the 'pointer' to the top of the program, whereas the STOP command does not.

29.3 Accessing the Stack Registers

As already mentioned, the HP 35s has a four register stack. On occasion, it may be useful to access these registers via an approach other than simply scrolling them into the X and Y stack registers.

29.3.1 Accessing the Stack Registers (Via the Equation List)

If you press EQN followed the R↓ key, you will see the following:

X Y Z T one of which will be underlined.

If you continue to press the R↓ key the underscore will move.

Displayed below the above letters will be the content of that particular stack register.

Let's say the underscore is below the Y. If you now press ENTER, you will see REGY appear in the equation list. Now press + (for addition) and then the R↓ key again.

This time, underline the X and press ENTER.

You will now see REGY + REGX_ blinking cursor. Press ENTER to complete the expression, followed by C to escape the equation list.

Now as an example, let's put an 8 in the Y stack register, and a2in the X stack register: Type 8; press ENTER; type 2 (Don't press ENTER); then EQN; then ENTER.

You will see 10 (the sum of the Y and X stack registers) appear in the lower display line.

This is a very trivial example, i.e., $REGY + REGX$, but keep in mind that the equation could contain other variables, constants, functions, etcetera. I should add that these are typed as if in algebraic mode.

29.3.2 Accessing the Stack Registers (Via Programming Mode)

A simple program equivalent to the one above would be as follows:

GTO . .

 PRGM

 LBL M Program M for example

REGY + REGX (See Note Below)

Note: For the above: Press EQN (won't show) then the R↓ key then underscore Y then ENTER then + (plus sign) then R↓ key then underscore X then ENTER and finally another ENTER to complete the line.

 RTN

C to escape program entry mode.


Now to run the above:

Type 8 then press ENTER then type 2 (Don't press ENTER).

Then press XEQ M ENTER

You should now see 10, the sum of the X and Y stack registers in the lower display. (If I had used registers Z and T, this example might have been more impressive.)

29.4 LASTX -- Last value of the X stack register (and something interesting that happens)

 **LASTX** recalls the last X stack register value that was actually 'acted upon' (as opposed to just 'appearing' there).

For example, type the following commands:


2

ENTER

1/X 1/X key Now shows 0.5000 in X register.

9 Some other random value entered just to simulate activity.

ENTER

 **LASTX** Shows 2.000 and not 0.5000 or 9.

Now with 9 in Y and 2 in the X registers.

Y^X Shows 81.000 (i.e., 9²)

Now with 0.5 in Y and 81 in X register.

 LASTX Shows 2.000 and not 81.

3 a random number typed (NO ENTER) Now with 2 in Y and 3 in X.

+ Now shows 5

 LASTX Shows 3.

At this point, I thought I knew what was going on: It was the LASTX of the number that was actually entered. However, look at the following:


5 Typed

1/X 1/X key

0.2 Now shows in X register.


1/X 1/X key

5 Now shows in X register.

 LASTX Shows 0.2 in the X register. Not a number that you entered.

Moral of the story: Be sure to check what actually happens in your application when using LASTX. See page 2.8 and page B.6 in the HP 35s User's Guide.

29.5 UNDO

 **UNDO** is useful during programming or when in equation mode. If you delete a program line or an equation (or part of an equation), UNDO can get it back.


UNDO is also useful if you have entered a number, but then mistakenly delete it; or if you cleared some digits and didn't mean to, by pressing UNDO repeatedly, the digits will reappear.

30.0 Miscellaneous

30.1 Unlabeled Program Stored Above the Other Programs

Note: You can put an additional (twenty-seventh) unlabeled program at the top of the program memory.

To enter the program, (while the X and Y stack registers are visible), press GTO . . (This puts the programming pointer at the top of the programming memory.)

Next press,  **PRGM** to go into programming mode. (You will see PRGM TOP in the lower window.) You won't be able to label the program, so skip that step.

 **INPUT D** Some command, just as an example.

 **VIEW D** Some command, just as an example.

 **RTN** A return marks the end of the program.

 **PRGM** To exit programming mode.

To execute this program (while you are seeing the X and Y stack registers), you press GTO . . This takes you to the top of the programming memory, and then press R/S to run the program.

However, this program has limitations. One is that it is not possible to use commands such as GTO, in that the statement lines have no letter prefix. (That is, rather than looking like A001, the lines are just 0000. In addition, with no label, other programs will not be able to branch to this program. (See HP 35s User's Guide, pages 13.2 and 13.4)

Note: This program will also not show up when you choose 2PGM on the MEM menu.

30.2 To 'Grab' Stack Contents and Store As a Variable

Let's say you want to grab the Z stack register contents during a program, and then store the contents in variable A. We'll label the program P.

 **LBL P**

REGZ Press EQN, then the R↓ key underlining Z, then ENTER, then ENTER (to escape).

 **STO A**

 **VIEW A**

 **RTN**

Note: The result will also be put in the X stack register, pushing the previous contents up.

See the HP 35s User's Guide B-7.

30.3 Comment on Evaluating Equations

(An Example from the HP 35s User's Guide Page 6.11):

The example shows: $X^2 + Y^2 = R^2$

Let's say you want to solve for R when $X = 2$ and $Y = 3$

If you use the EQN followed by ENTER as in the Method 1 (Section 22.2.1 above), you will get an answer of 13. That is, it doesn't exactly solve for R but rather it solves for the right side of the equation. The routine will ask for X, then Y, and then R. When asked for R, enter 0 and press R/S, and 13 will appear in the X stack register.

Note: If you entered $R^2 = X^2 + Y^2$ (i.e., with R^2 on the left), you will get an answer of -13 (minus 13).

If you use the EQN followed by SOLVE as in the Method 2 (Section 22.2.2 above), you will be asked for which variable to solve. Enter R.

You will then be asked for values of X and Y. Enter 2 and 3.

In this case you will get an answer of 3.6056. That is, the actual value of R (the square root of 13).

Note: This approach works regardless of whether R^2 is on the left or right side.

If you use the EQN followed by XEQ as in the Method 3 (22.2.3 above), you will get an answer of 13. That is, it doesn't exactly solve for R but rather it solves for the right side of the equation.

The routine will ask for X, then Y, and then R. In the first two cases, enter 2, then 3. When asked for R, enter 0 and press R/S, and 13 will appear in the X stack register.

Again, if you reverse the equation, you will get -13.

Comment: Method 2 with EQN followed by SOLVE is probably the method of choice in this particular case. SOLVE actually gives you an 'identified' answer, i.e., "R = 3.6056".

30.4 How to Step through a Program

Sometimes it is useful to step through a program, one line at a time. This is particularly useful if the program is not working properly.

To do this, go to the top of the desired program, for example B, as follows:


GTO B _ _ _ ENTER (001 will be automatically added.)

Press and hold the down arrow key (v symbol below MEM).

This will show each line as it is executed, pausing for any input or view responses.

30.5 The Integer Function Key


This function can be very useful. Below is the description of the menu's six options.


 **INTG** Menu: All results are based on the value in the X stack register.

1SGN 2INT÷


3Rmdr 4INTG


5FP 6IP


 **INTG** 1 If < 0 returns -1 ; if 0 returns 0 ; if > 0 returns 1 . The returns go into the X stack register. This command is interesting, in that it gives three outcomes.

 **INTG** 2 Gives the result of integer division. That is: $7 / 2 = 3$

 **INTG** 3 Gives the remainder from division. That is: $10 / 3 = 1$

 **INTG** 4 Gives greatest integer. That is, $\text{INTG}(8.9) = 8$

 **INTG** 5 Gives fractional part. That is: $\text{INTG}(8.9) = 0.9$

 **INTG** 6 Gives the integer portion of number.
That is: $\text{IP}(5.3) = 5$

30.6 “Culled” Material I Didn’t Have the Heart to Cull

30.6.1 Relationship between the XEQ statement and the RTN statement

This is a program (actually only the essence of a program) that originally puzzled me.

The program was basically supposed to VIEW A (11), then VIEW B (222), then VIEW W(999), and then stop. However, after viewing A and B, it was viewing W (999) twice.

First, to get things underway, store 111 in A, 222 in B, and 999 in W.

The program consists of two consecutive XEQs and some VIEW statements.

A001 LBL A

A002 VIEW A 111

A003 XEQ B001

A004 XEQ W001

W001 LBL W

W002 VIEW W 999

W003 RTN

B001 LBL B

B002 VIEW B 222

B003 RTN Go back to A004

Note: The RTN at W003 goes back to W001 (i.e., the line following the XEQ the first time through), and goes to the PRGM TOP the second time. My error, which caused the program to VIEW W (999) twice, was failing to put a RTN after the second XEQ.

In regards to the RTN statement: It seems to be the case (although I have no access to the code), that the 'default' action of a RTN command is to return to the program top. However, if the program contains an XEQ statement, that statement appears to issue a one-time instruction to the RTN it 'activates', telling it to return to the line following the XEQ.

30.6.2 To Bring the T Stack Register into the X Stack Register

Note: The example use the T stack register, but it also works for the Y, or Z stack register.

LBL X

REGT EQN, then R↓, then underline T, then ENTER (See note below.)

RTN

Note: Rather than the REGT, one may substitute three R↓ statements.

LBL X

R↓

R↓

R↓

RTN

In both cases, the contents of the T stack register will end up in the X stack register.

30.6.3 When Entering Numbers into an Executing Program

You have two choices after entering a number: If you press ENTER (before pressing R/S), the number goes into both the X and Y stack registers. However, if you just press R/S, the number enters only the X stack register.

30.6.4 The Stack is the Key to Efficiency

Watch the stack during complex program testing. By knowing what is in the stack (primarily the X and Y registers), a number of commands can usually be omitted.

31.0 Conditional Statement Summary

The X?0 decision rule is: Do [next command] if true. Otherwise skip the next command. See section 14.0

The X?Y decision rule is: Do [next command] if true. Otherwise skip the next command. See section 17.0

The FS? decision rule is: Do [next command] if true [that the flag is set]. If the flag is not set (i.e., is cleared), the following command is skipped.

Note: For the above, the next command is executed if the query is true.

ISG The 'memory tool' is: Increment [the start value index] and Skip [the next command] if [the index is] Greater Than [the terminal value]. See section 15.1


DSE The 'memory tool' is: Decrement [the start value index] and Skip [the next command] if [the index is] Less than or Equal to [the terminal value]. See section 19.0

Note: For both of the above (and to state things a little differently), the next command is processed as part of the loop. The 'escape' skips over the next command.

32.0 A Peculiarity about the Stack and the Recall (RCL)

Since the stack is at the heart of most operations, it is worth knowing of any peculiarities. The 'legitimacy', for lack of a better word, of a zero in the X stack register affects what happens during a recall.

For example:

Store 123 in the A variable: Type 123, then 

Press C to clear the 123 from the X stack register.

Type 456 and press ENTER Note: The 456 is now in both the X and Y stack registers.

Press the C to clear the X stack register. Note: The 456 is still in the Y stack register.

Observe that the 'default' 0.0000 is in the X stack register and will NOT move up into the Y stack register when we recall the value of the A variable in the next step. It will simply be replaced.

RCL A

Now 123 is in the X stack register and the 456 is in the Y stack register.

In other words, if the 'default' 0.000 is in the X stack register it is not moved up to the Y stack register. However, if you enter a 0 in the X stack register, it does move up when the recall takes place.

33.0 Suggestion

Since one can't include comments in the program itself to document what's going on, I recommend keeping a small notebook that includes:

Purpose of the program; date written; program label; meaning of the single letter variable 'names'; any 'housekeeping' required (variables to be cleared, flags to be set or cleared, required contents of any registers or variables entered at the start, prompts and order of any data to be entered, description of output (views, stack contents); LN =; CK=; and, of course, one could copy (write out) the program statements themselves, including line numbers.



ISBN 9781548763374



90000



9 781548 763374