HP 41 C / HP 41 CV / HP 41 CX

PANAME ROM

USER MANUAL
## FOREWORD

1

## WARNING

4

## FINDAID

5

## AID

6

*Appendix C*

7

## ID

8

## OUT

9

## OUTAX

10

## OUTCR OUTLF OUTLFX

11

## OUTSPX

12

## OUTXB OUTYBX

13

## OUTa OUTaX

15

## RCLSEL

16

### 82163 FUNCTIONS GROUP

17

CLEAR CLEARO CSRDN CSRHX

18

CSRL CSROFF CSRON CSRR CSRVX

19

CSRUP CTYPE HOME SCRLDN SCRLUP

20

SCRLX XYTAB

21

*Appendix V*

22

### 82162 FUNCTIONS GROUP

23

8BIT ESCAPE PARSE CLBUF

24

UNPARSE TABCOL

25

### 82905 FUNCTIONS GROUP

26

BELL CHARSET FFEED FORMLN GRAPHX

27

MODE SKIPOFF SKIPON TEXTLEN

28

VSPAC

29

*Appendix P*

30

*Roman8 Characters*

31

## MINIPLOTTER

32

AXIS

33

*LDIR *LTYPE *MOVE *PLREGX RDRAW RESET

35

REVLF REVLFX RMOVE SETORG

36

BACKSP BACKSPX BOX *COLOR *CSIZE *DRAW *HOME

37

*LABEL

38

STATUS

39

*Appendix T2*

40

## UTILITIES

41

/MOD

42

AD–LC

44

ALENG

45

ANUM

46

ANUMDEL

47

APPX

49

AROT

50

ATOXL ATOXR ATOXX

51

BLDPT

53

BRKPT

55

CHFLAG

57
CLINC  59
COLPT  60
GETRGX  61
LC-AD  63
LINPT  64
NOP  65
POSA  66
PSIZE SIZE?  67
READEM  68
RG  69
RG+  RG*  RG/  70
RG+Y  RG+Y  RG/Y  73
RGAX  75
RGCOPY  77
RGINIT  79
RGNb  81
RGSUM  82
RGVIEW  84
SORT  87
STO>L  89
SUB$  91
TF55  93
VKEYS  94
WRTEM  95
X<>F  96
X...NN?  98
Y/N  99

Appendix ON  100
FOREWORD

When the HP-41 was released, in 1979, it represented a great improvement in its field; it was able to receive, process, and display alphabetical strings. A real dialogue was now possible between the user and the calculator, this new flexibility was even improved by some sound possibilities.

In fact these advantages were only the most visible improvements. The alphanumeric keyboard induces other new and powerful capacities.

The first possibility is very useful in program mode. Instructions are not in code, but in plain language. Now "specialised machine language" is not adequate for the HP-41, "specialised assembly language" is better. In fact, for this calculator, HEWLETT-PACKARD has developed an advanced language, Forth-like, which places the HP-41 in a class of its own.

The comprehension of this potentialities induces the way one will use the HP-41.

THIS LANGUAGE IS AN INTERPRETED ONE.

The instructions inserted in the calculator memory are not directly intelligible to the microprocessor. Prior to execution they must be translated into a succession of micro-instructions, which are understandable by the HP-41 chip. This deciphering operation is called interpretation.

A SYMBOLIC LANGUAGE FOR THE HP-41.

A computer spends most of its time searching its memory to transfer informations from one place to another. This information can be transformed in the process, but it is not always necessary. To execute these transfers the microprocessor must know the data origin and destination, both are absolute addresses.

There are two types of informations:

- data: numeric values or characters strings.
- instructions: whose sequence represents a program.

At the machine language level all these informations are numbers. But the common users are very different from a microprocessor, they find it easier to remember words than numbers or instructions sequences: programmers prefer symbols to numbers. So the microprocessor must link, one way or another, the symbol and the address to get a given information. It can use catalogs which can be compared to a directory, where a telephone number is found using a surname. A language is characterized by its degree of symbolism.

THE HP-41 IS MODULAR.

In the world of micro-computers the HP-41 is one of the few machines which can contain an undetermined number of programs. They can be created, edited, erased, and (if you own a mass storage device) saved or loaded independently.
At this physical independence, is opposed a logical dependence. Any program can call a group of instructions belonging to another program. This sequence has only to begin by an alphabetical label : LBL "X..." and end by RTN or END.

Therefore it is possible to divide a complicated program, in a sequence of easier ones (and so on) according to the valuable principles of top-down programming. The problems of data handling are left to lower levels subroutines, while the logical sequence of the program is clearly visible at the higher levels. This programming technique has numerous advantages:

- It becomes very difficult to make mistakes while conceiving the different stages of a program. If it happens, it is easy to correct it because the mistake is quickly located within a small number of instructions.

- It is also very easy to test each subroutine to see if the output is consistent with a given input.

- Finally, some subroutines happen to be so useful that one wish to have an assembly language version of them. This module is a good illustration of it: most functions included in the PANAME ROM, where first created as subroutines in user language. The conception of the array handling techniques and of most functions were done in 1982.

PROGRAMMING THE HP-41.

The HP-41 has three programming levels.

- Programs are, in fact, a sequence of routines with tests in between. Always designed for a specific purpose, the program illustrates the strategical importance of the programmer's ability. A program must be understandable when reading it, and it must include its documentation.

- Routines, represent the tactical side of programming. A routine should be short, fast, memory saving, and modify as few variables as possible. It performs a specific task.

They are general enough to be used several times in a program, even in different programs. So, they should always be in memory. A standardisation of programming techniques saves time and efforts. If a routine meets all these requirements it can be considered as a new function for the HP-41 language: an illustration of the last quality of this language: its capacity of evolution.

- Assembly language functions. They are the elements of the language itself. A function should be general, even more than a routine. The two authors of this module provide us with a coherent group of more than 120 new functions.

FIRST CONCEPT: PERIPHERALS HANDLING.

Using the functions in this module is realizing what simplifications they offer when dealing with peripheral. Either using video or printer functions, a lot of time is saved when running a program or creating it. Plain language instructions instead of escape sequences, represents the same enhancement than reading "SIN" instead of "31 04". "CLEAR" is better than "27 ACCHR 69 ACCHR": it is more intelligible, and it works in trace mode. From this point of view the "PANAME" ROM is a great enhancement and it solves problems which had virtually no solution before.
ANOTHER CONCEPT: ARRAYS HANDLING.

How many times did we hear that the HP-41 was not designed for array handling. Now, every one will see how the use of the stack, that we have recommended, prepared us to these new functions. The rise of to FORTH language will give another evidence of this conception. Those lucky enough to use RPN logic will be prepared for FORTH, which perhaps, will replace BASIC.

We hope that all HP-41 programmers will appreciate the power of the "PANAME" ROM.

PHILIPPE DESCAMPS
WARNING

The PANAME ROM includes new functions for the HP-41, and they induce lots of different applications. But, as for other functions of the HP-41, either original ones or new ones, Users with their wide range of applications, are the only ones, who can show the interest of these functions.

This module would have been impossible without our PPC club, because it facilitates the exchange of solutions: it is very useful to share with other people knowledge and programming skills.

For the moment the PANAME ROM has a user manual, which has been made as clear and precise as possible. But we are aware of our limits. Therefore, we have decided to write, with everybody's help, a document following the spirit which was at the origin of the PPC Module.

If you are interested in a Solution Book for the PANAME ROM, you can put your name down with J.J. DHENIN, BCMW 2 bis rue N. HOUEL 75005 PARIS. When the solution book is ready you will be notified.

How do we intend to write this collective book?

Everyone will find unclear points in the original manual. We hope you will send us written questions. It will be better if a new redaction of these points is also proposed. As a matter of fact, we are so accustomed to these new functions that we are unable to evaluate the difficulties faced by a new user. So you are the only ones who can help us to enhance this manual.

Finally, examples are the best explanation, especially when the manual is not written in author native language. So we hope you will send us your own applications, short if possible.

According to your suggestions and to your work, we will be able to send you a new and better document in the future.

Happy programming.
FINDAID (FIND by Accessory ID) allows the HP-41 to find the location on the loop, of a device specified by its accessory ID. This function is the complement of the function FINDID in the HP82160A HPIL ROM, which allows searching for a device according to its device ID.

For greater ease-of-use, the FINDAID function can also locate a device of a specific class. A device class representing devices whose AID is in a specific range. For instance, the "Mass Storage" device class regroups all devices with an AID between 16 and 31. For the different class definitions refer to Appendix C.

INSTRUCTIONS FOR FINDAID

FINDAID allows the HP-41 to search a device of a given class or type which you specify with a number in the X-register. To specify a class, put in the X-register a negative number, which absolute value corresponds to the device class. Refer to Appendix C to set the number corresponding to each device class.

The FINDAID function starts its search at the primary device.

- If the specified device is found, its HP-IL address is returned to the X-register.
- If the search is unsuccessful 0 is returned to the X-register.

The old value of the X-register is always saved in LASTX.

EXAMPLE

1) To find the first printer on the loop, put -32 (Class number for "printer") in the X-register, and execute FINDAID.

2) Application program for FINDAID : 'FNDAIDN'

FNDAIDN searches the loop for the Nth device of a given class or AID.

To use FNDAIDN :
- Put N in the Y-register.
- Put the device class number or AID in the X-register. (As with FINDAID.)
- The result is left in the X-register; all other stack registers, including LASTX, are destroyed.

FNDAIDN listing :

```
LBL "FNDAIDN" RCLSEL STO 00 ST/ X
LBL 01 SELECT RCLSEL X≠Y? GTO 02 R^ R^ FINDAID X=0?
GTO 03 X<> L ISG L NOP LASTX DSE Z GTO 01 DSE X GTO 03
LBL 02 CLX LBL 03 RCL 00 SELECT RDN END
```

Related functions :

- HP-IL ROM: FINDID, SELECT, AUTOIO, MANIO.
- PANAME ROM: AID, ID, RCLSEL.
AID returns the Accessory ID of the primary device. The Accessory ID is an integer in the range 0 to 255 which identifies the device type.

For instance, the Accessory ID of the HP82162A thermal printer is 32. If the primary device is a HP82162A printer, the AID function returns 32 to the X register.

AID INSTRUCTIONS

The AID function returns to the X register an integer representing the Accessory Identity of the primary device. To compute the AID number of a device, refer to the description of the HPIL message "Sent Accessory Identity" in the device owner’s manual.

If the primary device has no Accessory Identity, the error message NO RESPONSE is displayed.

Related functions

EXTENDED I/O MODULE : FINDAID, ID
HPIL ROM : FINDID, SELECT, AUTOIO, MANIO
PANAME ROM : RCLSEL
Appendix C

For every type of accessory, this list gives the name, identity range and number related to a device type.

To find a device of a given type, store the type identifier into the X register and execute FINDAID.

<table>
<thead>
<tr>
<th>Type</th>
<th>AID</th>
<th>Type identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>controller</td>
<td>0 to 15</td>
<td>- 1</td>
</tr>
<tr>
<td>mass storage</td>
<td>16 to 31</td>
<td>- 16</td>
</tr>
<tr>
<td>printer</td>
<td>32 to 47</td>
<td>- 32</td>
</tr>
<tr>
<td>display</td>
<td>48 to 63</td>
<td>- 48</td>
</tr>
<tr>
<td>interface</td>
<td>64 to 79</td>
<td>- 64</td>
</tr>
<tr>
<td></td>
<td>80 to 95</td>
<td>- 80</td>
</tr>
<tr>
<td>graphic device</td>
<td>96 to 111</td>
<td>- 96</td>
</tr>
<tr>
<td></td>
<td>112 to 127</td>
<td>- 112</td>
</tr>
<tr>
<td></td>
<td>128 to 143</td>
<td>- 128</td>
</tr>
<tr>
<td></td>
<td>144 to 159</td>
<td>- 144</td>
</tr>
<tr>
<td></td>
<td>160 to 175</td>
<td>- 160</td>
</tr>
<tr>
<td></td>
<td>176 to 191</td>
<td>- 176</td>
</tr>
<tr>
<td></td>
<td>192 to 207</td>
<td>- 192</td>
</tr>
<tr>
<td></td>
<td>208 to 239</td>
<td>- 224</td>
</tr>
<tr>
<td></td>
<td>240 to 255</td>
<td>- 240</td>
</tr>
</tbody>
</table>
ID (Device ID) returns the Device ID of the SELECTed device. The Device ID is an alphanumeric string which identifies the device. Generally the ID string indicates the device manufacturer and reference.

For instance, the Device ID of the HPIL-RS232 interface is "HP82164A". If the SELECTed device is the HPIL-RS232 interface, the ID function returns "HP82164A" to the ALPHA register.

INSTRUCTIONS FOR ID

The ID function returns the Device ID of the SELECTed device to the ALPHA register. To get the ID string of a device, refer to the description of the HPIL message "Sent Device Identity" in the device owner's manual.

If the SELECTed device has no device identity, the error message NO RESPONSE is returned.

RELATED FUNCTIONS:

HPIL ROM: FINDID, SELECT, AUTOIO, MANIO
PANAME ROM: AID, FINDAID, RCLSEL
Easy keyboard input of functions beginning with OUT is possible thanks to the [OUT] function. This function is very useful when it is assigned to a key. For instance assign OUT to the [LN] key.

Keystrokes: [ASN][ALPHA][O][U][T][ALPHA][LN]
Put the calculator in USER mode, then execute or program a function whose name begins with OUT, for instance OUTAX.

Keystrokes: [OUT][[LN] key][ALPHA][A][X][ALPHA]

Without the OUT function the keystrokes would have been [XEQ][ALPHA][O][U][T][A][X][ALPHA], so you save 3 keystrokes every time you type a function beginning with OUT.

**OUT INSTRUCTIONS**

1) Assign OUT to a key and set the calculator to USER mode.

2) To execute or program a function beginning with OUT, strike:

[OUT] (It is assigned to a key)

[ALPHA]

...characters of the function name without the first three one.

...(for instance YBX for the OUTYBX function).

[ALPHA]
OUTAX performs one or several OUTAs functions, it sends the ALPHA register contents to the SELECTed device. The absolute value of the X register indicates the number of OUTAs to be performed.

If flag 17 is cleared, an End Line sequence is added to the ALPHA string each time. (End Line : characters CR and LF, decimal code 13 and 10).

If Flag 17 is set, the ALPHA string is sent several time without any separation character.

INSTRUCTIONS FOR OUTAX

The string to be sent several times must be put in the ALPHA register, the number of repetitions in the X register, and Flag 17 must be set or cleared according to the required effect (as mentioned above) then execute OUTAX.

EXAMPLE:

To draw a line of 40 ".*" on a HP82905B printer, use the following sequence. (The printer must be the SELECTed device)

".*" SF 17 40 OUTAX ADV

RELATED FUNCTIONS:

Any function beginning with OUT.

HPIL ROM : MANIO, SELECT are used to select the device.
- **OUTput Carriage Return** -

**OUTCR**

OUTCR sent a CR character to the SELECTed device. (Carriage Return : decimal code 13).

- **OUTput Line Feed** -

**OUTLF**

OUTLF sends a LF character to the SELECTed device. (Line Feed : decimal code 10).

- **OUTput Line Feeds by X** -

**OUTLFX**

OUTLFX Sends one or several LF characters to the SELECTed device. (Line Feed : decimal code 10). The number of characters is specified by the absolute value of the X register. ( 0<= X <= 999 ).

**INSTRUCTIONS FOR OUTLFX.**

Put the number of LF characters to be sent in the X register and execute OUTLFX.
OUTSPX (OUTput SPaces by X) sends the number of space characters (decimal code 32) specified by the absolute value of the X-register, (0\leq X\leq 999), to the primary device.

INSTRUCTIONS FOR OUTSPX

Put in the X-register the number of space characters to be sent to the primary device, and execute OUTSPX.

EXAMPLE

Numerous printers have no tabulation functions. The OUTSPX function replaces it quite well. For instance, the program OUTAT sends to the printer an alphabetical string of a given length L, representing the string in the ALPHA register followed, if necessary, by several space characters. If the string in the ALPHA register is longer than L, it is shortened to the first L characters (1).

The string length is limited to 24 characters because of the size of the ALPHA register.

OUTAT use:

- Put the string length (L) in the X-register.
- Type the string in the ALPHA register.
- Execute OUTAT.

The OUTAT program destroys registers X, T, LASTX and sets flag 17.

Note: L must be a positive integer number.

Listing of OUTAT:

```
LBL "OUTAT" ALENG X\geq Y? GTO 01
LBL 02 SF 17 OUTA OUTSPX RTN
LBL 01 DSE Y NOP CLX 1 E2 / SUB$
CLX GTO 02 END
```

N.B.: The text is left-justified. To print a text right-justified just swap OUTA and OUTSPX in OUTAT.

(1): In this case, the ALPHA register is modified by OUTAT.
- Sent a character by its decimal code -

OUTXB sends to the primary device, the character, whose decimal code is specified by the absolute value of the X-register. This value must be in the range 0-255.

INSTRUCTIONS FOR OUTXB

Put the decimal code of the character in the X-register, and execute OUTXB.

EXAMPLE

To sent to the printer the character "\" (Decimal code 92), use the sequence : 92 OUTXB.

- Send a character, several time, by its decimal code -

OUTYBX sends to the primary device, one or several times, a character whose decimal code is specified by the absolute value of the Y-register. The absolute value of the X-register specifies the number of characters to sent.

Restrictions : 0<=ABS(X)<=999 and 0<=ABS(Y)<=255.

INSTRUCTIONS FOR OUTYBX

Put in the Y-register the decimal code of the character and the character count in the X-register, then execute OUTYBX.

EXAMPLES

1) To send 20 "\" characters to the printer (Decimal code 39), use the sequence :

39 ENTER^ 20 OUTYBX.

2) 'PRNBLZ' (PRint Number with Leading Zeroes)

This program prints numbers with leading zeroes. The entry conditions are :

- The X-register holds the number to be printed.
- The Y-register holds the length of the printing field (maximum number of digits).
- Select the display format.
- Execute PRNBLZ.

If the printing field cannot hold the formatted number, it is field with "*" characters.

After execution, registers X, Y, LASTX and ALPHA are lost.
Listing of PRNBLZ:

LBL "PRNBLZ" CLA ARCL X X<0? XEQ 00
CLX ALENG X>Y? GTO 01 - 48 X<>Y OUTYBX OUTA RTN
LBL 00 CLX ATOXL OUTXB RTN
LBL 01 CLX 42 X<>Y OUTYBX END
OUTA with 7th bit set -

OUTa works like OUTA except that the 7th bit of every character sent is set. Therefore 128 is added when the character code is smaller than 128 with two important exceptions: LF and CR characters which are automatically sent after an ALPHA string when flag 17 is clear. (CR: carriage return, decimal code 13. LF: line feed decimal code 10).

INSTRUCTIONS FOR OUTA

Put in the ALPHA register the string to be sent, set or clear flag 17 (see above), execute OUTa.

EXAMPLES

1) To display a string in "reverse video" mode on the HP82163 video interface, you have to add 128 to each character code before sending the string to the interface. The OUTa function does it automatically. So to display a string in reverse video, select the interface as the primary device, put the string in the ALPHA register and execute OUTa. Flag 17 enables or disables the sending of an "End-of-line" sequence.

2) Some printers can automatically underline if you add 128 to the code of the character code to be underline. The OUTa function makes this operation easier with such printers.

3) There are two ways to use special characters on the HP82905B:
   - Using the secondary character set mode, which give new meanings to the codes 32 to 127.
   - Using characters with codes higher than 127.

The second method is very easy using the OUTa function.

- **OUTa with repetition by X** -

OUTaX executes several times the OUTa function (refer to it). The absolute value of the X-register specifies the number of OUTas to be performed. If flag 17 is clear, an "End-of-line indicator (CR and LF, decimal codes 13 and 10) is sent after each string. If flag 17 is set the string is sent several times without any other character.

INSTRUCTIONS FOR OUTaX

Put the string in the ALPHA register, the number of OUTas to be performed in the X-register, set or clear flag 17 (see above), then execute OUTaX.

EXAMPLE

To display a line of 16 "-**" strings in "reverse video" on the HP82163 video interface (which has to be the primary device), use the following sequence:

"-**" SF 17 16 OUTaX
RCLSEL (ReCaLl SELected address) returns in the X-register, after stack lift if it is enabled, the HP-IL address of the primary device. This address is an integer number. The RCLSEL function also checks the loop integrity (for device in standby mode it has the same effect as the PWRUP function, refer to the HP82160A HPIL ROM manual). There is a difference between the "EXTENDED I/O ROM" RCLSEL function and the "PANAME ROM" one: The "PANAME ROM" RCLSEL function can return a value, different from the last address specified by the SELECT function. This happens when the SELECT function has been executed with an address greater than the number of devices on the loop; in this case the address returned by RCLSEL is 1. This characteristic is useful in programs with a routine executed once for every device on the loop. A test between the SELECT address and the address returned by RCLSEL will check if all devices have been tested. Refer to the programs LOOP in the Example section of AID and ID, and FNDAIDN in the example section of FINDAID, to see how this method is used.

INSTRUCTIONS FOR RCLSEL

Execute RCLSEL; an integer number, which represents the address of the primary device is returnrd to the X-register as specified above.

EXAMPLE

RCLSEL can be used to save the primary device selection at the beginning of a program, which might modify it, and to restore it upon program termination. Use RCLSEL STO nn at the beginning and RCL nn SELECT at the end.
82163 FUNCTIONS GROUP

This group of functions will make the HP 826163 video easier to use. A full control of the video interface is possible without escape sequences or control characters. For instance to clear the screen or to move the cursor down, CLEAR and CRSDN (CuRSor DowN) are used.

For all these functions the primary device must be the video interface. For the different ways to select a device refer to the functions FINDAID (in this manual) and FINDID (in the HPIL ROM HP82160A owners manual).

In AUTOIO mode, if the primary device has a device identity other than 48 (standard video interface) an AID ERR error message is displayed.

However in MANIO mode, this error checking is not performed. So it is possible to use these functions with video interfaces, such as the Mountain Computer MC00701A (AID 50), PAC-TEXT, or KRISTAL(*) MINITEL interface.

For further informations on escape sequences, refer to Appendix V.

* KRISTAL, Chemin des Clos Zirst 38240 MEYLAN (FRANCE), is an HP ICC.
- Clear the display -

CLEAR clears the display, sets the cursor to position (0,0) and selects the replacement cursor.(*)

INSTRUCTIONS FOR CLEAR

Execute CLEAR.

EXAMPLE

ESC E, which is sent by the function CLEAR, is the reset sequence of the HP82905B printer. So CLEAR can be used to reinitialize this printer, but it must be performed in MANIO mode because the Accessory ID of the HP 82905B is 33.

- Clear the display from the cursor -

CLEARO clears the display, starting from the cursor and down to the end of the display. Cursor type and position are unchanged.

INSTRUCTIONS FOR CLEARO

Execute CLEARO.

(*) This is not true for all non-HP video interface.

- Move cursor down -

CSRDN

CSRDN (CurSoR DowN) moves the cursor one position down. If the cursor is on the bottom line of the display, the cursor is not moved.

- Move cursor Horizontally by X -

CSRHX

CSRHX (move CurSoR Horizontaly by X) moves the cursor horizontally. The absolute value of X specifies the number of characters of the move and its sign the direction:

- For X<0, CSRHX performs (-X) CSRLs (moves the cursor left by (-X) characters).
- For X>=0, CSRHX performs X CSRRs (moves the cursor right by X characters).

For instance -1 CSRHX is equivalent to CSRL and 1 CSRHX is equivalent to CSRR.

INSTRUCTIONS FOR CSRHX

Put in X the number corresponding to the desired move, then execute CSRHX.
- Move the cursor to the left -

CSRL (CurSoR Left) moves the cursor one position to the left. If the cursor is at position (0,0), it is not moved.

- Suppress the cursor -

CSROFF (CurSoR OFF) suppresses the cursor. The cursor is not visible until the next execution of CLEAR or CSRON or the next interface initialization (Power on or HPIL message -DCL- or -SDC-).

- Display the cursor -

CSRON (CurSoR ON) switches the cursor on. It can be switched off using CSROFF.

- Move the cursor to the right -

CSRR (CurSoR Right) moves the cursor one position to the right. If the cursor is at the end of a line, the cursor is sent to the beginning of the next line, except if it is at the end of the last line, in which case it is not moved.

- Move the cursor down according to X -

CSRVX (move CurSoR Vertically by X) moves the cursor vertically. The absolute value of X specifies the number of lines of the move and its sign the direction:
- For X<0, CSRVX performs (-X) CSRUPs (moves the cursor up by (-X) lines).
- For X>=0, CSRVX performs X CSRDNs (Moves the cursor down by X lines).

INSTRUCTIONS FOR CSRVX

Put in X, the number corresponding to the desired move, then execute CSRVX.
- Move the cursor up -

CSRUP

CSRUP (CurSoR UP) moves the cursor one position up. If the cursor is on the top line of the display, it is not moved.

- Select the type of cursor -

CTYPE

CTYPE (Cursor TYPE) selects the type of cursor according to the value of X:
- For X=0, selects the "insertion" cursor (blinking arrow);
- For X=1 or -1, selects the "replacement" cursor (blinking block).

INSTRUCTIONS FOR CTYPE

Put in X the value specifying the desired type of cursor and execute CTYPE. Beware that when using the Video interface Mountain Computer MC00701A, the selection of the insertion cursor (Blinking underline) selects neither "character insertion" mode nor "line insertion" mode.

- Put the cursor at the upper left position of the display -

HOME

HOME moves the cursor to position (0,0).

- Scroll the display down -

SCRLDN

SCRLDN (SCRoIL DowN) scroll the display on line down. (So the bottom line disappears and a new line appears at the display top.)

- Scroll the display one line up -

SCRLUP

SCRLUP (SCRoIL UP) scroll up the display by one line. (So the top line of the display disappears and a new line appears at the bottom of the screen.)
**INSTRUCTIONS FOR SCRLX**

Put in $X$, the number corresponding to the desired scrolling, and execute SCRLX.

**INSTRUCTIONS FOR XYTAB**

Put in $X$ the column number, in $Y$ the line number and execute XYTAB.
Appendice V

Sequences sent to the primary device by the HP82163 FCNS group.

"ESC" represents the escape character, decimal code 27.

<table>
<thead>
<tr>
<th>Function(s)</th>
<th>Sequence</th>
<th>Characters codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>ESC E</td>
<td>27 69</td>
</tr>
<tr>
<td>CLEARO</td>
<td>ESC J</td>
<td>27 74</td>
</tr>
<tr>
<td>CSRDN, CSRVX for X&gt;=0</td>
<td>ESC B</td>
<td>27 66</td>
</tr>
<tr>
<td>CSRL, CSRHX for X&lt;0</td>
<td>BS</td>
<td>08</td>
</tr>
<tr>
<td>CSROFF</td>
<td>ESC &lt;</td>
<td>27 60</td>
</tr>
<tr>
<td>CSRON</td>
<td>ESC &gt;</td>
<td>27 62</td>
</tr>
<tr>
<td>CSRR, CSRHX for X&gt;=0</td>
<td>ESC C</td>
<td>27 67</td>
</tr>
<tr>
<td>CSRU P, CSRVX for X&lt;0</td>
<td>ESC A</td>
<td>27 65</td>
</tr>
<tr>
<td>CTYPE for X=0</td>
<td>ESC Q</td>
<td>27 81</td>
</tr>
<tr>
<td>CTYPE for X=1 or -1</td>
<td>ESC R</td>
<td>27 82</td>
</tr>
<tr>
<td>HOME</td>
<td>ESC H</td>
<td>27 72</td>
</tr>
<tr>
<td>SCRLDN, SCRLX for X&gt;=0</td>
<td>ESC T</td>
<td>27 84</td>
</tr>
<tr>
<td>SCRLUP, SCRLX for X&lt;0</td>
<td>ESC S</td>
<td>27 83</td>
</tr>
<tr>
<td>XYTAB</td>
<td>ESC % {c} {l}</td>
<td>27 37 col ln</td>
</tr>
</tbody>
</table>
82162 FUNCTIONS GROUP

This group of functions makes easier the operation of the HP82162A Thermal Printer. You will be able to use every features of this printer, even those not described in the manual.

These features are:

- Two different character sets;
- A "parse" mode;
- A absolute dot-level tabulation function, independent from any data already in the printer buffer;
- The possibility to obtain status information from the printer.

These functions work on the first HP82162A printer on the loop starting from the primary device. If no HP82162A printer is found on the loop, the error message "NO 82162" is displayed. The STATUS function of the "PANAME ROM" is the only exception to this rule; refer to this function for further details.
- Select "8 bit mode" -

8BIT

8BIT selects "8 bit mode", which validates the HP41 character set. This mode is automatically selected when a specific printer function is used. (One of the -PRINTER 2E functions of the HP-IL ROM.) This function is useful only if the HP82162A printer is used with non-specific printer functions, such as OUTA or OUTYBX.

- Select "Escape" mode -

ESCAPE

ESCAPE selects the "escape" mode, which activates the ASCII (non HP-41) character set. In this mode, you cannot use specific printing functions to send characters to the printer because they automatically select the "8 bit mode". However some applications may require the use of the ASCII character set. The ESCAPE function enables the use of this set, but printing must be done with the OUTA function, or related functions, which only send characters to the primary device. Beware that in this case, the primary device must be the printer, even though this is not necessary with specific printer functions such as PRA.

-Select "Line feed on space" mode -

PARSE

PARSE selects parse mode, which enables automatic word-wrap at end of lines. A line feed is performed by the printer on the space, when the next word cannot be printed completely on the current line.

- Clear the buffer -

CLBUF

CLBUF returns the printer to power on status:

- The printer head is at the right;
- The printer buffer is empty;
- selected modes are: "escape", single width, uppercase letters, left justification, line-feed on the 24th character.

This function is mainly used to clear the printer buffer of any data, it is the only way to do it.
UNPARSE enables the special mode selected by UNPARSE.

TABCOL enables an absolute tabulation on the dot level as opposed to SKIPCOL, which permits a relative tabulation.

Using TABCOL, it is easy to print an output with several columns (Only two columns with FMT !).

INSTRUCTION FOR TABCOL

Put the column number (0 to 167) in X and execute TABCOL.

EXAMPLE

To print the following chart:

A = 123.00 FF
B = 23.95 FS
C = 1115.70 FB

You can use the following sequence:

```
FIX 2  CLBUF  "A=" ACA 28 TABCOL 123 ACX 91 TABCOL "FF" ACA PRBUF
   "B=" ACA 28 TABCOL 23.95 ACX 91 TABCOL "FS" ACA PRBUF
   "C=" ACA 28 TABCOL 1115.7 ACX 91 TABCOL "FB" ACA PRBUF
```
82905 Functions Group

This group of functions will make the HP82905B 80-column printer much easier to use. Thanks to them you can completely control the printer without knowing escape sequences or control characters normally needed to perform a specific task. These functions permits an easier writing or reading of programs using the various modes of the HP82905B.

For all these functions the printer must be the primary device. Refer to FINDAID (in this manual) or to FINDID (in the HP82160A HPIL ROM manual) to find how to select a given device.

In AUTOIO mode, if the primary device does not have an AID of 33, the error message AID ERR is displayed.

However, this check is not performed in MANIO mode. So they can be used with other printers using the same escape sequences or control characters.

For more informations on sequences sent by these functions, refer to Appendix P.
- Beep signal -

**BELL**

BELL rings the "bell" of the printer for one second. This function can be used to call the attention of the user.

- Character set selection -

**CHARSET**

CHARSET selects the primary character set if X=0, and the secondary one if X=1. Refer to the HP82905B printer User's Manual for informations on both character sets.

- Form feed -

**FFEED**

FFEED sends to the printer a "form feed" command, which sets the printer to the top of the next page. Beware that you must position the paper correctly and set the number of lines per page (using FORMLEN) prior to using FFEED.

- Page length -

**FORMLEN**

FORMLEN defines the number of lines per page (It is related to the paper's physical form length and line spacing selected with VSPAC).

The absolute value of X indicates the number of lines, which must be in the range 1-128. At power on or after reinitialization with the CLEAR function (refer to this function for further information) the default line count is 66.

- Graphic output -

**GRAPHX**

GRAPHX indicates to the printer that the next X bytes received are binary data, not characters, each value representing a dot column. Refer to the printer User's Manual to find the relations between data sent and printer output (Graphic mode section).

The absolute value of X represents the number of bytes to be considered as graphic data.
- Printing mode -

**MODE**

MODE selects the printing mode according to the absolute value of X:

<table>
<thead>
<tr>
<th>X value</th>
<th>Mode</th>
<th>Nb char./line</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>Expanded</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>Compressed</td>
<td>132</td>
</tr>
<tr>
<td>3</td>
<td>Compressed-Expanded</td>
<td>66</td>
</tr>
<tr>
<td>9</td>
<td>Bold</td>
<td>80</td>
</tr>
</tbody>
</table>

You can combine modes "0" and "1" or "2" and "3"; other combinations give unpredictable results.

If X has other values than "0", "1", "2", "3", or "9", the error message DATA ERROR is displayed.

- Disables perforations skip -

**SKIPOFF**

SKIPOFF cancels the SKIPON function.

- Enables perforation skip-

**SKIPON**

SKIPON enables perforations skip mode on the printer. When this mode is on, the printing of the last text line of a page generates a form feed: the paper is set to the beginning of the next page. (the number of text line per page is selected with the TEXTLEN function). So nothing will ever be printed across two pages.

Perforations skip mode is off, at power on or upon execution of the CLEAR function. (Refer to this function for further information).

- Text length -

**TEXTLEN**

TEXTLEN sets the number of number of lines in the "text area" equal to the absolute value of X. This number must be in the range 1 to the number of line per page (selected with FORMLEN). At power on or after executing the CLEAR function, the default is 60 text (useable) lines per page.
- Vertical spacing -

**VSPAC**

VSPAC selects the vertical spacing in lines per inch according to the absolute value of X. This number must be 6,8,9,12,18,24,36 or 72. Any other value will cause a DATA ERROR.
Appendix P

Sequences sent to the primary device by the 82905 FNCS functions.

- ESC represents the escape character, decimal code 27.
- {#} symbolises the ASCII representation of a number and {parm} the related character codes.

<table>
<thead>
<tr>
<th>Function(s)</th>
<th>Sequence</th>
<th>Codes</th>
<th>Thinkjet</th>
</tr>
</thead>
<tbody>
<tr>
<td>BELL</td>
<td>BEL</td>
<td>07</td>
<td></td>
</tr>
<tr>
<td>CHARSET for X=0</td>
<td>SI</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>CHARSET for X=1 or -1</td>
<td>SO</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>FFEED</td>
<td>FF</td>
<td>12</td>
<td>FF</td>
</tr>
<tr>
<td>FORMLEN</td>
<td>ESC &amp;l (#) P</td>
<td>27 38 108 (parm)</td>
<td>FL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>GRAPHX</td>
<td>ESC *b (#) G</td>
<td>27 42 98 (parm)</td>
<td>71</td>
</tr>
<tr>
<td>MODE</td>
<td>ESC &amp;k (#) S</td>
<td>27 38 107 (parm)</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 Normal (80 c/l), 1 expanded (40 c/l), 2 compressed (142 c/l), 3 expanded-compressed (71 c/l)</td>
<td></td>
</tr>
<tr>
<td>SKIPOFF</td>
<td>ESC &amp;l0L</td>
<td>27 38 108 48 76</td>
<td>skipoff</td>
</tr>
<tr>
<td>SKIPOFF</td>
<td>ESC &amp;l1L</td>
<td>27 38 108 49 76</td>
<td>skipon</td>
</tr>
<tr>
<td>TEXTLEN</td>
<td>ESC &amp;l (#) F</td>
<td>27 38 108 (parm)</td>
<td>textlen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>VSPAC</td>
<td>ESC &amp;l (#) D</td>
<td>27 38 108 (parm)</td>
<td>vspac</td>
</tr>
<tr>
<td></td>
<td></td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>CHAR.</td>
<td>DEC.</td>
<td>HEX.</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>ctrl@</td>
<td>0 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlA</td>
<td>0 01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlB</td>
<td>0 02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlC</td>
<td>0 03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlD</td>
<td>0 04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlE</td>
<td>0 05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlF</td>
<td>0 06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlG</td>
<td>0 07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlH</td>
<td>0 08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlJ</td>
<td>0 09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlK</td>
<td>0 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlL</td>
<td>0 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlM</td>
<td>0 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlN</td>
<td>0 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlO</td>
<td>0 14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlP</td>
<td>0 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlQ</td>
<td>0 17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlR</td>
<td>0 18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlS</td>
<td>0 19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlT</td>
<td>0 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlU</td>
<td>0 21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlV</td>
<td>0 22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlW</td>
<td>0 23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlX</td>
<td>0 24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlY</td>
<td>0 25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrlZ</td>
<td>0 26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrl</td>
<td>0 27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrl</td>
<td>0 28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrl</td>
<td>0 29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrl</td>
<td>0 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctrl</td>
<td>0 31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Roman-8 Characters (ASCII)**

<table>
<thead>
<tr>
<th>CHAR.</th>
<th>DEC.</th>
<th>HEX.</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>32</td>
<td>20</td>
</tr>
<tr>
<td>&quot;</td>
<td>33</td>
<td>21</td>
</tr>
<tr>
<td>#</td>
<td>34</td>
<td>22</td>
</tr>
<tr>
<td>$</td>
<td>35</td>
<td>23</td>
</tr>
<tr>
<td>%</td>
<td>36</td>
<td>24</td>
</tr>
<tr>
<td>&amp;</td>
<td>37</td>
<td>25</td>
</tr>
<tr>
<td>'</td>
<td>38</td>
<td>26</td>
</tr>
<tr>
<td>(</td>
<td>39</td>
<td>27</td>
</tr>
<tr>
<td>)</td>
<td>40</td>
<td>28</td>
</tr>
<tr>
<td>*</td>
<td>41</td>
<td>29</td>
</tr>
<tr>
<td>+</td>
<td>42</td>
<td>2A</td>
</tr>
<tr>
<td>.</td>
<td>43</td>
<td>2B</td>
</tr>
<tr>
<td>/</td>
<td>44</td>
<td>2C</td>
</tr>
<tr>
<td>0</td>
<td>45</td>
<td>2D</td>
</tr>
<tr>
<td>1</td>
<td>46</td>
<td>2E</td>
</tr>
<tr>
<td>2</td>
<td>47</td>
<td>2F</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>49</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>51</td>
<td>33</td>
</tr>
<tr>
<td>7</td>
<td>52</td>
<td>34</td>
</tr>
<tr>
<td>8</td>
<td>53</td>
<td>35</td>
</tr>
<tr>
<td>9</td>
<td>54</td>
<td>36</td>
</tr>
<tr>
<td>:</td>
<td>55</td>
<td>37</td>
</tr>
<tr>
<td>;</td>
<td>56</td>
<td>38</td>
</tr>
<tr>
<td>&lt;</td>
<td>57</td>
<td>39</td>
</tr>
<tr>
<td>=</td>
<td>58</td>
<td>3A</td>
</tr>
<tr>
<td>&gt;</td>
<td>59</td>
<td>3B</td>
</tr>
<tr>
<td>?</td>
<td>60</td>
<td>3C</td>
</tr>
<tr>
<td>!</td>
<td>61</td>
<td>3D</td>
</tr>
<tr>
<td>`</td>
<td>62</td>
<td>3E</td>
</tr>
<tr>
<td>_</td>
<td>63</td>
<td>3F</td>
</tr>
<tr>
<td>a</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>A</td>
<td>65</td>
<td>41</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
<td>42</td>
</tr>
<tr>
<td>C</td>
<td>67</td>
<td>43</td>
</tr>
<tr>
<td>D</td>
<td>68</td>
<td>44</td>
</tr>
<tr>
<td>E</td>
<td>69</td>
<td>45</td>
</tr>
<tr>
<td>F</td>
<td>70</td>
<td>46</td>
</tr>
<tr>
<td>G</td>
<td>71</td>
<td>47</td>
</tr>
<tr>
<td>H</td>
<td>72</td>
<td>48</td>
</tr>
<tr>
<td>I</td>
<td>73</td>
<td>49</td>
</tr>
<tr>
<td>J</td>
<td>74</td>
<td>4A</td>
</tr>
<tr>
<td>K</td>
<td>75</td>
<td>4B</td>
</tr>
<tr>
<td>L</td>
<td>76</td>
<td>4C</td>
</tr>
<tr>
<td>M</td>
<td>77</td>
<td>4D</td>
</tr>
<tr>
<td>N</td>
<td>78</td>
<td>4E</td>
</tr>
<tr>
<td>O</td>
<td>79</td>
<td>4F</td>
</tr>
<tr>
<td>P</td>
<td>80</td>
<td>50</td>
</tr>
<tr>
<td>Q</td>
<td>81</td>
<td>51</td>
</tr>
<tr>
<td>R</td>
<td>82</td>
<td>52</td>
</tr>
<tr>
<td>S</td>
<td>83</td>
<td>53</td>
</tr>
<tr>
<td>T</td>
<td>84</td>
<td>54</td>
</tr>
<tr>
<td>U</td>
<td>85</td>
<td>55</td>
</tr>
<tr>
<td>V</td>
<td>86</td>
<td>56</td>
</tr>
<tr>
<td>W</td>
<td>87</td>
<td>57</td>
</tr>
<tr>
<td>X</td>
<td>88</td>
<td>58</td>
</tr>
<tr>
<td>Y</td>
<td>89</td>
<td>59</td>
</tr>
<tr>
<td>Z</td>
<td>90</td>
<td>5A</td>
</tr>
<tr>
<td>{</td>
<td>91</td>
<td>5B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>92</td>
</tr>
<tr>
<td>}</td>
<td>93</td>
<td>5D</td>
</tr>
<tr>
<td>~</td>
<td>94</td>
<td>5E</td>
</tr>
<tr>
<td>_</td>
<td>95</td>
<td>5F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAR.</th>
<th>DEC.</th>
<th>HEX.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>96</td>
<td>60</td>
</tr>
<tr>
<td>a</td>
<td>97</td>
<td>61</td>
</tr>
<tr>
<td>b</td>
<td>98</td>
<td>62</td>
</tr>
<tr>
<td>c</td>
<td>99</td>
<td>63</td>
</tr>
<tr>
<td>d</td>
<td>100</td>
<td>64</td>
</tr>
<tr>
<td>e</td>
<td>101</td>
<td>65</td>
</tr>
<tr>
<td>f</td>
<td>102</td>
<td>66</td>
</tr>
<tr>
<td>g</td>
<td>103</td>
<td>67</td>
</tr>
<tr>
<td>h</td>
<td>104</td>
<td>68</td>
</tr>
<tr>
<td>i</td>
<td>105</td>
<td>69</td>
</tr>
<tr>
<td>j</td>
<td>106</td>
<td>6A</td>
</tr>
<tr>
<td>k</td>
<td>107</td>
<td>6B</td>
</tr>
<tr>
<td>l</td>
<td>108</td>
<td>6C</td>
</tr>
<tr>
<td>m</td>
<td>109</td>
<td>6D</td>
</tr>
<tr>
<td>n</td>
<td>110</td>
<td>6E</td>
</tr>
<tr>
<td>o</td>
<td>111</td>
<td>6F</td>
</tr>
<tr>
<td>p</td>
<td>112</td>
<td>70</td>
</tr>
<tr>
<td>q</td>
<td>113</td>
<td>71</td>
</tr>
<tr>
<td>r</td>
<td>114</td>
<td>72</td>
</tr>
<tr>
<td>s</td>
<td>115</td>
<td>73</td>
</tr>
<tr>
<td>t</td>
<td>116</td>
<td>74</td>
</tr>
<tr>
<td>u</td>
<td>117</td>
<td>75</td>
</tr>
<tr>
<td>v</td>
<td>118</td>
<td>76</td>
</tr>
<tr>
<td>w</td>
<td>119</td>
<td>77</td>
</tr>
<tr>
<td>x</td>
<td>120</td>
<td>78</td>
</tr>
<tr>
<td>y</td>
<td>121</td>
<td>79</td>
</tr>
<tr>
<td>z</td>
<td>122</td>
<td>7A</td>
</tr>
<tr>
<td>{</td>
<td>123</td>
<td>7B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>124</td>
</tr>
<tr>
<td>}</td>
<td>125</td>
<td>7D</td>
</tr>
<tr>
<td>~</td>
<td>126</td>
<td>7E</td>
</tr>
<tr>
<td>_</td>
<td>127</td>
<td>7F</td>
</tr>
</tbody>
</table>
Several Miniplotters can be used with the HP-41. TANDY, CANON ... have the same mechanism and the same command set. Of course, the miniplotter should be interfaced with the HP-IL loop with the HP82166A converter.

Several companies produce HP-IL interfaced miniplotters, or a converter and parallel interface, which can be used with such miniplotters.

The main features of those miniplotters are:

- 4 colors. The printhead is in fact a cylinder of 4 mini ball pens. The color can be changed, either by program or by a switch, during plotting.

- 11.4 cm in roll paper. It is possible to draw, or plot on the length of the paper, so one can print large charts.

- Horizontally you can print 80 c/l.

- Of course, these miniplotters can be used with other HP-IL controllers such as the HP-75, HP-85, HP-71. So these devices will not be obsolete soon.
Axis drawing

AXIS draws several kind of axis on the mini-plotter.

\[...\]

INSTRUCTIONS FOR AXIS

AXIS uses four parameters, which must be on the stack before executing the function:
T: half-length of a tick
Z: vertical distance between two ticks
Y: horizontal distance between two ticks
X: number of ticks

The axis is drawn from the current pen position, and the direction depends on the values in the Y and Z registers. However, ticks are always either vertical or horizontal according to the axis inclination from the horizontal (X direction): under 45 degrees, ticks are vertical, over they are horizontal.

The parameter in T makes the drawing of charts easier. For instance with arrays.

Example: The following program draws a chart with 2 lines and C columns. Each column has a width of W and each line an height of H. To use it, only type XEQ'CHART' and answer the questions. (Input the value and R/S).

```
01 LBL "CHART"
02 "HP82166" GP-IO Converter identification.
03 FINDID Search the address of the mini-plotter.
04 SELECT Select the miniplotter.
05 RESET Reinitialization.
06 "Nb. COL. ?" Input the number of columns (C).
07 PROMPT R00= Number of columns.
08 STO 00
09 "COL. WIDTH ?" Input the columns width.
10 PROMPT
```
11 STO 01
12 *
13 "HT. LINE ?"
14 PROMPT
15 STO 02
16 ST+ X
17 CHS
18 X<>Y
19 0
20 ENTER^  
21 BOX
22 RCL 02
23 CHS
24 0
25 *MOVE
26 RCL 02
27 0
28 RCL 01
29 RCL 00
30 AXIS
31 END

R01= Columns width.
First chart dimension.

Input the line height.
R02= Height of each line.
There are 2 lines, the 2nd dimension is 2*X.
The displacement will be down!

BOX uses the 4 parameters T, Z, Y and X.

Starting position.

Drawing of inside lines.

Will print:

XEQ "CHART"
Nb. COL. ?
4.0000 RUN
COL. WIDTH ?
100.0000 RUN
HT. LINE ?
50.0000 RUN
- Writing direction in graphic mode -

*LDIR

*LDIR specifies the writing direction for *LABEL. There are four possibilities:
0 to the right, 1 down, 2 to the left, 3 up.

- Line Type -

*LTYPE

*LTYPE (Line TYPE) specifies one of the 16th possible line types of the miniplotter.
The value of the X-register is considered modulo 16. The line type will be used with the
DRAW and RDRAW functions.

- Move pen up -

*MOVE

*MOVE moves the pen, without plotting, to the (X,Y) coordinates.

- Broken line plotting -

*PLREGX

*PLREGX (PLot REGisters by X) joins the points, whose coordinates are specified by
successive registers.
The X-register contains a bbb,eee pointer; the integer part (bbb) specifies the register
storing the first coordinate of the first point, the fractional part (eee) specifies the register
storing the second coordinate of the last point. If in the succession of data the calculator
finds one or several ALPHA strings the pen skip to the next cordinate (numeric data)
without plotting (*MOVE), then it resumes plotting (DRAW).

- Relative Drawing -

RDRAW

RDRAW (Relative DRAWing) draws a line up to the position (x,y) relative to the current
position of the pen.

- Reinitialization -

RESET

RESET moves the pen to the left margin and selects Text mode.
- Reverse line feed -

**REVLF**

REVLF (REVersin Line Feed) moves the pen one line upward.

- Reverse line feed by X -

**REVLFX**

REVLFX (REVerse Line Feed by X) moves the pen upward of the number of lines specified by the absolute value of the X-register.

- Relative MOVEment -

**RMOVE**

RMOVE (Relative MOVEment) moves to the (x,y) position relative to the current pen position.

- Set origin -

**SETORG**

SETORG (SET ORigin) sets the current pen position has the origin (0,0).
- BACKSPACE -

BACKSP moves the pen one character backward

- BACKSPACE BY X -

BACKSPX moves the pen backward by the number of characters specified by the absolute value of the integer part of the X-register.

- Box drawing -

BOX draws a rectangle, whose 2 opposite angles have the coordinates: (x1, y1) and (x2, y2), with T=y2, Z=x2, Y=y1, X=x1.

- Color selection -

*COLOR selects one of the four colors according to the value of the X-register.

- Character size -

*CSIZE (Character SIZE) selects the character size. The value of the X-register must be in the range 0-63.

- Draw a segment -

*DRAW draws a line segment from the current pen position to the (X, Y) coordinates.

- Sent pen to origin -

*HOME move pen to (0,0) coordinates.
- Print the ALPHA register -

*LABEL

*LABEL prints the ALPHA register. This function is useful because the drawing can be done in four directions in text mode: these four directions are specified by the *LDIR function.
- Recall printer status -

**STATUS**

STATUS returns to the Y-register an integer, which is the printer first status byte, and in the X-register an integer, which is the printer second status byte. The effect of STATUS on the stack depends on whether stack lift is enabled or not when the function is executed:

- If stack lift is enabled:

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: y</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: x</td>
</tr>
<tr>
<td>Y: y</td>
<td>Y: 1st status byte</td>
</tr>
<tr>
<td>X: x</td>
<td>X: 2nd status byte</td>
</tr>
</tbody>
</table>

- If stack lift is disabled:

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: z</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: y</td>
</tr>
<tr>
<td>Y: y</td>
<td>Y: 1st status byte</td>
</tr>
<tr>
<td>X: x</td>
<td>X: 2nd status byte</td>
</tr>
</tbody>
</table>

However, the LASTX-register is not modified.

The STATUS function has a specific feature: in MANIO mode, it returns to the X and Y registers two numbers, which specify the primary device status.

- If the primary device has no status bytes, STATUS returns 97 to the X and Y registers;

- If the primary device has only one status byte, STATUS returns the decimal representation of this byte to Y register, and returns 64 to the X-register;

- If the primary device has, at least, two status bytes, STATUS works with the primary device, as with the HP82162A printer in AUTOIO mode. Status bytes after the second one are ignored.

To compute the number, and definition of the status bytes of a device, refer to the description of the HP-IL message "SENT STATUS" in the device manual.

The appendix S1 gives the detailed definition of the two status bytes of the HP82162A printer.
APPENDICE T2

Minimum function set needed to use a 4 color mini-plotter with the PLOT FCNS functions group.

Refer to JPC n15 june 1984 for a description of the mini-plotter.

Representation convention:

- # represents a numeric character string, with an optional minus sign and no more than 4 digits (For instance: -230; 0024);

The syntax column specifies the signification of each parameters;

Control characters (Decimal values):
17: Select TEXT mode
18: Select GRAPHICS mode
11: Reverse line feed (Text mode only)
08: Backspace (Text mode only)

GRAPHICS mode instructions

Syntax Format Action

A A Initialization
H H Home (Position (0,0))
Mx,y M#,,# Move to position (x,y)
Dx,y D#,,# Drawing to position (x,y)
Rx,y R#,,# Relative move of (x,y)
Jx,y J#,,# Relative drawing of (x,y)
Pstring Pstring Printing of the characters string "string"
Lx L# Select line type x
Cx C# Select pen x (Change color)
Sx S# Select character size x
Qx Q# Select printing direction x (for P instruction only).

GRAPHICS mode functions

The mini-plotter instructions, which correspond to these functions, require the Graphics mode. So these functions set Graphic mode before executing the operation, and leave the mini-plotter in this mode after execution.

Functions that are not mode specific

These mini-plotter instructions must be executed in Graphic mode. So these functions set Graphic mode before executing the instruction. However these functions are oftenly used in Text mode. The user can control in which mode the mini-plotter is left after execution.
This function group has a wide range of applications:

- Character string manipulation;
- Manipulation of numeric and alphanumeric arrays (one or two dimensions);
- Numeric or alphanumeric sorting;
- Extended memory management (HP82180A XFUNCTIONS and HP82181A XMEMORY);
- Wide range of other applications...!
- Euclidian division -

/MOD (Divide MOD) computes the remainder and the quotient of an Euclidian division (i.e. with integers). It is an extension of the built-in [MOD] function.

EXAMPLE

- Calculation of the modulus and quotient of the division of 13 by 3.

<table>
<thead>
<tr>
<th>Input</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>13_</td>
</tr>
<tr>
<td>[ENTER^]</td>
<td>3 3_</td>
</tr>
<tr>
<td>[XEQ] &quot;/MOD&quot;</td>
<td>1,0000</td>
</tr>
<tr>
<td>[X&lt;&gt;Y]</td>
<td>4,0000</td>
</tr>
<tr>
<td>[LASTX]</td>
<td>3,0000</td>
</tr>
</tbody>
</table>

INSTRUCTIONS FOR /MOD

1) To compute the modulus and quotient of the division of Y by X.

2) [XEQ] "/MOD". The quotient and modulus of the division are returned respectively to the Y and X registers. The divisor is saved in the L-register, the dividend is lost. T and Z registers are left unchanged.

3) If the X-register contains 0, the calculator displays DATA ERROR.

STACK

Input:               Output:

T: t               T: t
Z: z               Z: z
Y: Dividend         Y: quotient
X: Divisor          X: Remainder
L: l               L: Divisor

APPLICATION PROGRAMS FOR /MOD

1) A fairly quick way to compute the decimals of the division of A by B when A<B and the last digit of B is 9:

LBL "DIV9"  10 / INT 1 + STO 01 RDN SF 21 LBL 01 RCL 01 /MOD VIEW Y 10 * + GTO 01 END

So to divide 153 by 209

153/209=0,732057...

2) [/MOD] can be used in a short subprogram as a small base conversion! This short program, "YBX", gives the digits of the new number; but in reverse order. X and Y must be integers.

That means: 1103 (DEC) = 2117 (OCT). This results can be check with the DEC and OCT functions.

N.B : If it is possible to get the divisor back with \(X<>Y\) \(\text{LASTX} \times +\) for a quotient >0 and with \(X<>Y\) \(X<0?\) \(\text{DSE X NOP} \times \text{LASTX} \times +\) for a quotient <0, it is impossible for a quotient equal to 0.
AD-LC returns the coordinates (line, column) of an array element from its address (Rnn) and the array pointer.

Example: Compute the coordinates of register 36 in the array A (Below), which array pointer 25,04405 is in R00. R25 = first array element, R44 = last array element.

<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line no 1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td>line no 2</td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td>line no 3</td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td>line no 4</td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
</tbody>
</table>

ARRAY A

Input: Display:
36 [ENTER] 36,0000 Input register no.
[RCL] 00 25,04405 Recall the array pointer.
[XEQ] "AD-LC" 2,00000 Column no2.
[RDN] 3,00000 Line no3.
[LAST X] 25,04405 The pointer is saved in L.

- INSTRUCTIONS FOR AD-LC

To get the line, column coordinates of an array element when you know the array pointer and the register address of this element: Input the register number, [ENTER^], array pointer, [XEQ] "AD-LC". The column number is returned in the X-register and the line number in the Y-register. The array pointer is saved in L, registers Z and T are unchanged.

STACK

Input: Output:
T: t T: t
Z: z Z: z
Y: Register no Y: line no
X: Array pointer X: Column no
L: L L: Array pointer

NOTA: This function does not check if the register is part of the array.
If registers X or Y contains an Alpha string, ALPHA DATA is displayed.
[ALENG] returns to the X-register the length of the current string in the ALPHA register.

Example 1: In a program, the HP-41 stops and waits for an ALPHA input. The string length is needed to store the string in several registers. An other solution is the RGAX function which is described in this manual.

INSTRUCTION FOR ALENG

Place in ALPHA the string, [ALENG] returns in the X-register the string length and the stack is lifted (if it is enable).

THE STACK

<table>
<thead>
<tr>
<th>Input:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: z</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: y</td>
</tr>
<tr>
<td>Y: y</td>
<td>Y: x</td>
</tr>
<tr>
<td>X: x</td>
<td>X: String length.</td>
</tr>
<tr>
<td>L: l</td>
<td>L: l</td>
</tr>
</tbody>
</table>

Application program for ALENG.

Example 2: The following routine upercases in the ALPHA register. It uses [ALENG] to provide a loop counter initially equal to the number of characters in the string (which must not contain null characters).

```
01 LBL "CAP"
02 ALENG  counts characters in ALPHA register.
03 LBL 00
04 ATOXL Places code of leading characters into X.
05 97  The lowercase letters are in the range 97 to 122.
06 X=Y? If not lower case character, go to [LBL] 01
07 GTO 01
08 CLX
09 122
10 X<Y? The character codes for uppercase letters
11 GTO 01 are determined by subtracting 32 from their
12 CLX  lowercase counterparts.
13 32  R^
14  R*
15 LBL 01
16 RDN
17 RDN  restores capitalized letter to ALPHA.
18 XTOAR branches if their is any character left.
19 RDN
20 DSE X
21 GTO 00 .END.
```
[ANUM] (Alpha to NUMber) searches the ALPHA register, from left to right, for a number (in ASCII form). The first number found is returned to the X-register.

Example: The ALPHA register contains the string: "PRICE: 1.234,50" read from an extended memory ASCII file. To extract the numeric value for further use: [XEQ] "ANUM" and the number is returned to the X-register.

INSTRUCTIONS FOR ANUM

1) The ANUM function searches for a numeric value in the ALPHA register string. If a number is found, it is returned to the X-register and flag 22 is set. If a number is not found, the X-register and flag 22 are left unchanged.

2) Numbers in the ALPHA register are processed according to the status of flags 28 and 29. If a number in the ALPHA register has a "-" sign, a negative number is returned to the X-register when the function is executed. Suppose that the ALPHA register contains the string of example 1:

<table>
<thead>
<tr>
<th>Flag 28</th>
<th>Flag 29</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>set</td>
<td>1,234.5000</td>
</tr>
<tr>
<td>set</td>
<td>clear</td>
<td>1,0000</td>
</tr>
<tr>
<td>clear</td>
<td>set</td>
<td>1,2345</td>
</tr>
<tr>
<td>clear</td>
<td>clear</td>
<td>1,2340</td>
</tr>
</tbody>
</table>

STACK

Input :          Output :
T:  t            T:  z
Z:  z            Z:  y
Y:  y            Y:  x
X:  x            X:  number found in ALPHA.
L:  l            L:  l
Search ALPHA number and delete -

ANUMDEL searches the current string in the ALPHA register, from left to right, for a number (in ASCII form) and returns to the X-register the value of the number. It also deletes all characters in the string from the start of the string to the last numerical character used.

Example 1: Suppose that the ALPHA register contains the string "PRICE: $1,234.5XYZ", to extract the numeric value for further use, [XEQ] "ANUMDEL" stores this number in the X-register; The ALPHA string is deleted up to "5" included.

INSTRUCTIONS FOR ANUMDEL

1) ANUMDEL searches the string in the ALPHA register for a numeric value. If a number is found, it is stored in the X-register and the string is deleted up to the last numerical character used to build the number.

2) If the ALPHA string contains more than one number separated by non-numeric characters, ANUMDEL uses only the first number. ANUMDEL is identical in operation to the ANUM function, except that the ANUM function does not alter the string. The HP-41 considers execution of ANUMDEL as a numeric entry, and sets flag 22, if a number is returned to the X-register. If the ALPHA string contains no numeric characters, ANUMDEL clears the ALPHA register but has no effect on the stack.

3) The characters "+", ",", ",", "E" (for exponent) are interpreted by ANUMDEL as numeric or non-numeric characters according to their context in the ALPHA string. An isolated "+", is not treated as a numeric character. A "+" or "," symbol immediately preceding, embedded in, or following a sequence of number digits will be interpreted exactly as if the symbols and numbers had been keyed into the X-register (with [CHS] representing "," and [CHS][CHS] representing "+".) For instance, ANUMDEL returns the value -3425 if executed when the ALPHA register contains the string "34-2+5".

The status of the numeric display control flags (flags 28 and 29) determines how the Alpha string is interpreted by ANUMDEL. For example, if flags 28 and 29 are both set, commas are treated as digit separators. But commas are considered as non-numeric if flag 28 is set and flag 29 is clear. Suppose that the Alpha register contains the string of example 1: "PRICE: $1,234.5XYZ". Set FIX 4 and execute ANUMDEL; the following table shows the result according to the setting of flags 28 and 29.

<table>
<thead>
<tr>
<th>Flag 28</th>
<th>Flag 29</th>
<th>X-Register</th>
<th>Modified Alpha String</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>set</td>
<td>1,234.5000</td>
<td>XYZ</td>
</tr>
<tr>
<td>set</td>
<td>clear</td>
<td>1,0000</td>
<td>,234.5XYZ</td>
</tr>
<tr>
<td>clear</td>
<td>set</td>
<td>1,2345</td>
<td>XYZ</td>
</tr>
<tr>
<td>clear</td>
<td>clear</td>
<td>1,2340</td>
<td>.5XYZ</td>
</tr>
</tbody>
</table>
STACK

Input : Output :

\[
\begin{align*}
T : t & \quad T : z \\
Z : z & \quad Z : y \\
Y : y & \quad Y : x \\
X : x & \quad X : \text{first numeric value found in ALPHA.} \\
L : l & \quad L : l
\end{align*}
\]

APPLICATION PROGRAM FOR ANUMDEL

Example 2: The HP 7470A Graphics Plotter can send on HP-IL an ASCII character string that describes the current pen position. The string contains three integer numbers separated by commas: X,Y,P. X is the pen’s x-coordinate; Y is the pen’s y-coordinate; P has a value of 1 if the pen is down, or 0 if the pen is up. Suppose that the plotter has sent the string "123,456,1" to the HP-41’s ALPHA register. You could use the following keystrokes to decipher the string:

Keystrokes | Display
---|---
[SF 28] | Ensures that a comma is not interpreted as a radix.
[ANUMDEL] | 123.0000 X-coordinate.
[ANUMDEL] | 456.0000 Y-coordinate.
[ANUMDEL] | 1.0000 Pen is down.

Example 3: ALPHA has the string "34/-2/5"

Keystrokes | Display
---|---
[CF 28] | 
[ANUMDEL] | 34.0000
[ALPHA] | -2/5
[ALPHA] [ANUMDEL] | -2.0000
[ALPHA] | /5
[ALPHA] [ANUMDEL] | 5.0000

This example shows that "/" and "#" are not considered as "+", ",", or ".".
APPX (APPend X) appends the integer part of the X-register to the right of the ALPHA register string.

Example: The result of an area computation is in the X-register: 1,225.7, and the message "AREA: " is in the ALPHA register, the APPX function appends the X-register value after the message, without rounding: ALPHA = "AREA: 1,225"

INSTRUCTIONS FOR APPX

1) [APPX] appends the integer part of the X-register to the left of the ALPHA register. [APPX] results depend on flags 28 and 29. The number is written as in FIX 0 mode, except that the decimal separator is not appended, and the number is not rounded. As for [ASTO] [APPX] does not beep, when its execution overflows the ALPHA register capacity.

2) If at the execution of [APPX], the X-register contains an alpha string, ALPHA DATA is displayed.
AROT (Alpha ROTate) rotates the ALPHA register string of the number of characters specified by the X-register.

Example: The ALPHA register contains the string "AROT". To display "TARO" then "ROTAX".

Input:

Display:

[ALPHA] AROT       AROT_
[ALPHA] 1 [CHS]    -1_
[XEQ] "AROT" [ALPHA] TARO
[ALPHA] 2          2_
[XEQ] "AROT" [ALPHA] ROTA

INSTRUCTIONS FOR AROT

[AROT] rotates the ALPHA register string of the number of characters specified by the value, modulo 24, of the X-register. The rotation is done to the left, if the X-register contains a positive number, and to the right if it is negative. (Refer to the appendix for further information on the effect of [AROT] on null characters). The execution of [AROT] does not modify the stack.

APPLICATION PROGRAMS FOR AROT

1) The [AROT] function can be used with the [ANUM] and [POSA] functions to get the number of repetition of a given string, or character, in the ALPHA register without destruction.

2) An operation on a device returns to the ALPHA register the following string "68.2 69.88" (a number, a space, a number). To extract separately two numbers to use them in a program, the following sequence can be used:

Input:

Display:

[CF] 28
[XEQ] "ANUM" 68.2000 Return the first number to the X-register.
32 32_ Space code.
[XEQ] "XTOAR" 32.0000 Add a space to the right of the ALPHA string.
[XEQ] "POSA" 4.0000 Search the first space in the ALPHA register.
[XEQ] "AROT" 4.0000 Rotate the string; ALPHA contains 69.88 68.2; Without a space ALPHA would contain 69.8868.2.
[XEQ] "ANUM" 69.8800 Return 69.88 to the X-register.
- Character transfer between ALPHA and X

- Transfer leftmost character of ALPHA to X -

\[ \text{ATOXL} \]

([ATOXL] (Alpha-TO-X Left) deletes the first character of ALPHA and returns its decimal code to the X-register.

- Transfer rightmost character of ALPHA to X -

\[ \text{ATOXR} \]

([ATOXR] (Alpha-TO-X Right) deletes the last character of ALPHA and returns its decimal code to the X-register.

- Transfer specified character of ALPHA to X -

\[ \text{ATOXX} \]

([ATOXX] (Alpha-TO-X by X) returns to the X-register the character of ALPHA specified by the value of the X-register. The ALPHA register is left unchanged.

**INSTRUCTIONS FOR ATOXL, ATOXR, ATOXX**

1) [ATOXL] deletes the leftmost character of the ALPHA register string and returns its decimal code to the X-register. If the first character is followed by one, or several null characters, the string is moved, to the left, up to the first non null character. If the ALPHA register is empty, [ATOXL] returns -1 to the X-register.

2) [ATOXR] deletes the rightmost character of the ALPHA register string, and returns its decimal code to the X-register. If the ALPHA register is empty, -1 is returned to the X-register.

3) [ATOXX] returns to the X-register the decimal code of the character, whose position in the string, is specified in the X-register. The ALPHA register is left unchanged.

A positive value in the X-register specifies a position in the ALPHA register string, starting form the first non null character. This first character is in position 0. This convention is the one used for the POSA function in the XFUNCTION module.

On the contrary, a negative number specifies an absolute position in the ALPHA register, it is independent from the string. Positions are considered from right to left, -1 for the rightmost position and -24 for the left most position. The following chart illustrates the [ATOXX] interpretation of the character positions.
If the X-register contains an Alpha string, ALPHA DATA is displayed.

Example:

In this example, the ALPHA register is completely represented, null characters at the left of the register are represented with horizontal marks, but they cannot be displayed by the calculator.

<table>
<thead>
<tr>
<th>Input</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ALPHA] DECAMETRE [ALPHA]</td>
<td>DECAMETRE</td>
</tr>
<tr>
<td>0 [XEQ] &quot;ATOXX&quot;</td>
<td>68.0000 Code of &quot;D&quot;</td>
</tr>
<tr>
<td>4 [XEQ] &quot;ATOXX&quot;</td>
<td>77.0000 Code of &quot;M&quot;</td>
</tr>
<tr>
<td>6 [CHS] [XEQ] &quot;ATOXX&quot;</td>
<td>65.0000 Code of &quot;A&quot;</td>
</tr>
<tr>
<td>10 [CHS] [XEQ] &quot;ATOXX&quot;</td>
<td>0.0000 Null character</td>
</tr>
</tbody>
</table>
Build a pointer

[BLDPT] (Build Pointer) builds a pointer bbb.eeeii if X>0 or an array pointer if X<0.

Example 1: A program has left in the Z-register the number of the first register of a set of data, in the Y-register the number of the last register of the set, and in the X-register the number of registers between two data. Z=25 Y=40 X=5

To compute the pointer: [XEQ] "BLDPT", [FIX] 5. X=25.04005 will give the address of R25, R30, R35, R40.

Example 2: A previous program has left in the Z-register the first register number of an array, in the Y-register the number of lines, in the X-register the number of columns: Z=25 Y=4 X=5.

To get the array pointer, [CHS] [XEQ] "BLDPT", X=25.04405

```
<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td>line no2</td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td>line no3</td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td>line no4</td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
</tbody>
</table>
```

ARRAY A

INSTRUCTIONS FOR BLDPT

1) To build a bbb.eeeii pointer:

- Put bbb in the Z-register;
- Put eee in the Y-register;
- Put ii in the X-register;
- Execute [BLDPT].
2) To build a bbb.eeecc array pointer; where bbb specifies the address of the first register used by the array, eee specifies the last register used by the array and cc the number of columns:

- Put bbb in the Z-register;
- Put the number of lines lll of the array in the Y-register;
- Put the number of columns cc of the array in the X-register, with a negative sign;
- Execute [BLDPT].

NOTA: If either the X, Y, Z register contains an Alpha string, ALPHA DATA is displayed.
The pointer is built with the absolute values of bbb and eee.

STACK:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: t</td>
</tr>
<tr>
<td>Z: bbb</td>
<td>Z: t</td>
</tr>
<tr>
<td>Y: eee</td>
<td>Y: t</td>
</tr>
<tr>
<td>X: ii</td>
<td>X: bbb.eeeii</td>
</tr>
<tr>
<td>L: l</td>
<td>L: eee</td>
</tr>
</tbody>
</table>

For X<0

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: t</td>
</tr>
<tr>
<td>Z: bbb</td>
<td>Z: t</td>
</tr>
<tr>
<td>Y: eee</td>
<td>Y: lll</td>
</tr>
<tr>
<td>X: cc</td>
<td>X: bbb.eeecc</td>
</tr>
<tr>
<td>L: l</td>
<td>L: lll</td>
</tr>
</tbody>
</table>
[BRKPT] (BReaK PoinTer) splits into its three components a bbb.eeeii pointer if X>0, or an array pointer if X<0.

EXAMPLES:

1) A program needs the elements of a bbb.eeeii pointer, where bbb is the first register of a set of data, eee is the last one and ii the number of values between two data in the set. X= 25.04005 specifies registers R25, R30, R35, R40 [XEQ] "BRKPT" returns Z=25, Y=40, X=5.

2) The array pointer is 25.04405, it specifies that the array begins at R25, ends at R44, and has 5 columns. The array number of lines is returned by : [CHS] [XEQ] "BRKPT". So Z=25 (1st register), Y=4 (number of lines), X=-5( number of columns).

<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td>line no2</td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td>line no3</td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td>line no4</td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
</tbody>
</table>

INSTRUCTIONS FOR BRKPT

1) To split up a bbb.eeeii pointer where bbb, in the range 0-999, is the first element of a loop or a vector; where eee, in the same range, is the last element; and where ii is the increment. One must check that the number in the X-register is positive, for instance with [XEQ] "ABS"; then [XEQ] "BRKPT" will return the integer part of the X-register to the Z-register, the first 3 digits of the decimal part to the Y-register, and the 4th and 5th digits of the decimal part to the Z-register.

The pointer is saved in LASTX.

2) To break a bbb.eeecc array pointer, where bbb is the register of the first element of the array, eee is its last register, and cc is the number of columns. One must insure that the number in the X-register is negative, for instance with [ABS] [CHS]; then [XEQ] "BRKPT" returns the first register (bbb) to the Z-register, the number of lines lll=(eee+1-bbb)/cc to the Y-register, and the number of columns (cc) to the X-register.

The array pointer is saved in LASTX.
Nota: If there is an Alpha string in the X-register, ALPHA DATA is displayed.

<table>
<thead>
<tr>
<th>STACK</th>
<th>for X&gt;0</th>
<th>For X&lt;0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>T: t</td>
<td>T: x</td>
<td>T: t</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: bbb</td>
<td>Z: z</td>
</tr>
<tr>
<td>Y: y</td>
<td>Y: eee</td>
<td>Y: y</td>
</tr>
<tr>
<td>X: bbb.eee</td>
<td>X: ii</td>
<td>X: bbb.eee</td>
</tr>
<tr>
<td>L: l</td>
<td>L: bbb.eee</td>
<td>L: l</td>
</tr>
</tbody>
</table>
- Load flag set -

**[CHFLAG]** (CHange FLAGs) restores the flag set that was current when the CHFLAG function was written in the program.

**Example**: At the beginning of a program, you want to be in DEGree mode, 3 digits ENG and with the 5 first flags \(0-4\) set.

While in RUN mode (PRGM indicator off) initialize the calculator as needed, then in PRGM mode [XEQ] "CHFLAG". It writes two lines in the program: the first line is CHFLAG, the second one is a 7 character string. When the program is executed the calculator is initialized to the needed state.

**INSTRUCTIONS FOR CHFLAG**

1) In RUN mode, initialize the calculator to the state needed by the program.

2) In PRGM mode, [XEQ] "CHFLAG" writes two lines, the first one is CHFLAG, the second one contains a seven character string, which represents the current flag set. This string begins with a configuration indicator. If this string is destroyed or replaced by a wrong one, CHFLAG execution will halt program execution and CHFLAG ERR will be displayed.

**STACK**:

[CHFLAG] execution does not affect the stack.

N.B: The ALPHA register is not modified by [CHFLAG]. The character string represents a set of flags, it is not for the ALPHA register.

One must not put a test instruction before CHFLAG as ISG or X=Y?.

E.g. : FS? 01 If the flag is set

CHFLAG Reinitializes the calculator.

'.....' Initialization string.

If the test is negative (Flag 01 clear) the ALPHA register is destroyed by the configuration string.

[CHFLAG] only saves flags 00 to 43.
F00 to F10: user's flags.

F11: Automatic execution of current program, at power on; or after reading one from mass memory.

F12 to F20: External device commands.

- **F12**: Double width.
- **F13**: Lower case letters.
- **F15** to **F16**: Printing mode of HP-IL printer.
  - 0 0: Manual
  - 0 1: Normal
  - 1 0: Trace
  - 1 1: Trace and stack printing.
- **F16**: CR-LF ignored
- **F17**: CR-LF ignored
- **F18**: CR-LF ignored
- **F19**: CR-LF ignored
- **F20**: Print enabled
- **F21**: Set by a numeric input.
- **F22**: Set by an alphanumeric input
- **F23**: Out of range ignored.
- **F24**: Error ignored
- **F25**: Beep on
- **F26**: User mode
- **F27**: Decimal separator type
- **F28**: Three digit groups separator
- **F29**: DMY mode of TIME module
- **F30**: MANIO mode on HP-IL module
- **F31**: ADROFF mode on EXTENDED I/O
- **F32**: Auto start enable (AUTOSTART/DUPLICATION ROM)
- **F33** to **F39**: Number of digits for FIX, SCI, ENG
- **F40** and **F41**: Display mode
- **F42** and **F43**: Angular mode
- Clear Increment -

[CLINC] (CLear INCrement) truncates the number in the X-register from the 4th digit of the decimal part.

Example: You want to compute the first and last registers of an array. The array pointer is saved in R00. Use the following sequence:

Keystrokes: Display

[RCL] 00 25.04405 Recall the array pointer
[XEQ] "CLINC" 25.04400
[XEQ] "INT" 25.00000 1st register
[LASTx] 25.04400
[XEQ] "FRC" 0.04400
[EE] 3 [*] 44.00000 Last register

INSTRUCTIONS FOR CLINC

[CLINC] replaces, in the X-register, any decimal digits after the 3rd one by 0. The old value is saved in LASTX.

STACK

Input: Output:

T: t T: t
Z: z Z: z
Y: y Y: y
X: bbb.eeeff X: bbb.eee
L: l L: bbb.eeffl

NOTA: If the X-register contains an Alpha string, ALPHA DATA is displayed.
Create column pointer -

[COLPT] (COLUMn PoinTer) returns a column pointer to the X-register, from the column number in the Y-register, and the array pointer in the X-register.

**Example**: to get the second column pointer of the array A, which pointer is in register 00.

**Keystrokes**:  

<table>
<thead>
<tr>
<th>Keystrokes</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.</td>
</tr>
<tr>
<td>[RCL]  00</td>
<td>25.04405 Recall pointer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>column no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
<th>ARRAY A</th>
</tr>
</thead>
<tbody>
<tr>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
<td></td>
</tr>
<tr>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
<td></td>
</tr>
<tr>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
<td></td>
</tr>
<tr>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
<td></td>
</tr>
</tbody>
</table>

**INSTRUCTIONS FOR COLPT**

1) Input the column number.

2) Put the array pointer in the X-register.

3) [XEQ] "COLPT" returns the column pointer to the X-register, and saves the array pointer in LASTX.

**STACK**:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: t</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: t</td>
</tr>
<tr>
<td>Y: Column NO</td>
<td>Y: z</td>
</tr>
<tr>
<td>X: bbb.eeei</td>
<td>X: bbb.eeei</td>
</tr>
<tr>
<td>L: l</td>
<td>L: bbb.eeei</td>
</tr>
</tbody>
</table>

**N.B.**: i'j'i=ji
[GETRGX] (GET ReGisters by X) copies in the registers specified by the X-register, the
data of the current Extended Memory file (file where the pointer is), starting from
pointer position, and according to the increment in the X-register.

**Example:** The pointer in the current file is on 10, 25.0440510 [XEQ] "GETRGX" copies
register 10, 20, 30,... from the X-Memory file, to registers 25, 30, 35,... in main memory.

**INSTRUCTIONS FOR GETRGX**

1) Set the current file pointer to the right position with [SEEKPT] or [SEEKPTA].

2) The number in the X register is a bbb.eeeiijj pointer, where bbb is the first main
memory register, eee the last main memory register where you want to copy the X-
Memory data set, ii is the increment for the main memory registers, and jj the increment
for the X-Memory registers.

3) [XEQ] "GETRGX" copies the registers from the X-Memory current file to the main
memory registers as specified by the pointer in the X-register.

**STACK:**

The stack is unchanged by [GETRGX].

**EXAMPLE**

The drawings below represents two arrays, the left one is in main memory, the right one
is in X-Memory. In each square is indicated the register number, and its value (a letter).

Set the X-Memory pointer to the first register to copy, with 12 [SEEKPT].

To copy the second column of the array B, in X-Memory to the 3rd column of array A in
main memory, put in the X-register the pointer of the 3rd column of array A (27,04205),
and add the increment for the X-Memory registers (03) as 6th and 7th decimal digits:

\[ X = 27.0420503 \]

27 = bbb 1st register in main memory.
42 = eee last register in main memory.
05 = ii increment for main memory registers.
03 = jj increment for X-Memory registers.

[XEQ] "GETRGX" copies the registers as specified by the pointer in the X-register; the
result is represented on the second drawing.
### BEFORE [GETRGX]

<table>
<thead>
<tr>
<th>col</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln 1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>ln 2</td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>ln 3</td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>ln 4</td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
</tr>
</tbody>
</table>

#### Array A main memory

#### Array B X-Memory

<table>
<thead>
<tr>
<th>col</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln no1</td>
<td>R11</td>
<td>R12</td>
<td>R13</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>ln no2</td>
<td>R14</td>
<td>R15</td>
<td>R16</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>ln no3</td>
<td>R17</td>
<td>R18</td>
<td>R19</td>
</tr>
<tr>
<td></td>
<td>g</td>
<td>h</td>
<td>i</td>
</tr>
<tr>
<td>ln no4</td>
<td>R20</td>
<td>R21</td>
<td>R22</td>
</tr>
<tr>
<td></td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
</tbody>
</table>

### AFTER [GETRGX]

<table>
<thead>
<tr>
<th>col</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln 1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>b</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>ln 2</td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>G</td>
<td>g</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>ln 3</td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>L</td>
<td>h</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>ln 4</td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>R</td>
<td>k</td>
<td>T</td>
<td>U</td>
</tr>
</tbody>
</table>

#### Array A main memory

#### Array B X-Memory

<table>
<thead>
<tr>
<th>col</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln no1</td>
<td>R11</td>
<td>R12</td>
<td>R13</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>ln no2</td>
<td>R14</td>
<td>R15</td>
<td>R16</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>ln no3</td>
<td>R17</td>
<td>R18</td>
<td>R19</td>
</tr>
<tr>
<td></td>
<td>g</td>
<td>h</td>
<td>i</td>
</tr>
<tr>
<td>ln no4</td>
<td>R20</td>
<td>R21</td>
<td>R22</td>
</tr>
<tr>
<td></td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
</tbody>
</table>
[LC-AD] (Line-Column-ADDress) returns the register number of an array element from its line number, column number, and array pointer.

Example: Register number of the element on line 2 and column 3 of array A, which array pointer (25.04405) is saved in R00.$

Keystrokes: Display:

[2] [ENTER] 2.0000 Input line number.
[3] 3_ Input column number.
[RCL] 00 25.04405 Recall array pointer.
[XEQ] "LC-AD" 32.00000 Register NO.

INSTRUCTIONS FOR LC-AD

To compute the register number of an array element, when you know its line number, column number, and array pointer: Input line number, [ENTER^], Column number, [ENTER^], array pointer. [XEQ] "LC-AD" returns the register number to the X-register, and saves the pointer in LASTX.

STACK:

Input: Output:

T: T T: T
Z: line no Z: T
Y: column no Y: T
X: Array Pointer X: register no
L: L L: Array pointer
- Create line pointer -

[LINPT] (LINe PoinTer) returns a line pointer to the X-register, given the line number in the Y-register and the array pointer in the X-register.

**Example**: To compute the registers used by the 2nd line of array A, whose array pointer is assumed to be found in register R00:

Keystrokes:  

```
[RCL] [0] [0] 25.04405 Recall array pointer.
[XEQ] "LINPT" 30.03400 2nd line pointer.
```

<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td>line no2</td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td>line no3</td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td>line no4</td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
</tbody>
</table>

**INSTRUCTIONS FOR LINPT**

1) Input the line number, whose pointer is needed.

2) Put the array pointer in the X-register.

3) [XEQ] "LINPT" returns the line pointer to the X-register and saves the array pointer in LASTX.

**STACK**:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: t</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: t</td>
</tr>
<tr>
<td>Y: line NO</td>
<td>Y: z</td>
</tr>
<tr>
<td>X: bbb.eeeii</td>
<td>X: bbb.eeeii</td>
</tr>
<tr>
<td>L: l</td>
<td>L: bbb.eeeii</td>
</tr>
</tbody>
</table>
- No Operation -

[NOP] (No OPeration) is used after a test, when the conditional goto is not used.

Example: To increment the X and Y registers in a loop.

The following sequence will do it:

ISG Y Increment the Y-register.
NOP No OPeration.
ISG X Increment the X-register,
GTO 03 and looped if higher.
- Position of an string in ALPHA -

[POSA] (POSition in Alpha) searches the ALPHA register, from left to right, for the character or string specified in the X-register.

**Example 1**: The string "ABCDEFGHIJKLMNOPQRSTUVWXYZ" is in the ALPHA register, what is the position of the 1st "D" character?

**Keystrokes**:

```
68 68_ "D" character code.
[XEQ] "POSA" 3.0000 1st "D" character position.
```

**Example 2**: [ALPHA] [CLA] DEF [ASTO] . X ABCDEFGHIJ [ALPHA] [XEQ] "POSA" X=3.00

**INSTRUCTIONS FOR POSA**

1) [POSA] searches the ALPHA register, from left to right, for the character or string specified in the X-register. The string can be specified either by giving a character code, or by storing the string or character in the X-register with [ASTO] [.]. X. If the calculator finds the string in the ALPHA register, it returns the 1st character position to the X-register.

2) Positions are considered from left to right and start with position 0. If the string or character appears several times in the ALPHA register, the calculator returns only the first position. If the string or the character does not exist in the ALPHA register, -1 is returned.

3) The string or character code is saved in LASTX.

**STACK**:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td></td>
</tr>
<tr>
<td>Z: z</td>
<td></td>
</tr>
<tr>
<td>Y: y</td>
<td></td>
</tr>
<tr>
<td>X: code or string</td>
<td>X: position in ALPHA</td>
</tr>
<tr>
<td>L: l</td>
<td>L: code or string</td>
</tr>
</tbody>
</table>
- MEMORY ALLOCATION FUNCTIONS -

- Programmable SIZE -

[PSIZE] (Programmable SIZE) allocates the number of data registers specified by the X-register.

- Number of data registers -

[SIZE?] returns to the X-register the number of data registers.

[SIZE?] and [PSIZE] can be used in the same program to change the number of data registers without loosing any data.

EXAMPLE :

01 ....
02 ....
.... Your program
....
07 SIZE? Return to the X-register the number of data registers.
08 125 The new program needs 125 data registers.
09 X>Y? The current number of data registers is in the Y-register.
09 X>Y? Does the program needs more data registers ?
10 PSIZE Change if necessary.
[READEM] (READ Extended Memory) copies from a mass memory file (HP82161A tape drive) the whole contents of X-Memory, which was previously saved in the file with the WRTEM function.

Example: To load the file "MAT3" from the tape.

Keystrokes: Display:

[XEQ] "EMDIR" DIR EMPTY Checks that the X-Memory is empty.
If two XMEMORY modules are plugged in, there are 600 registers available.

[ALPHA] "MAT 3" [ALPHA] 600.0000 File name in ALPHA.

[XEQ] "READEM" 600.0000 the files are loaded into X-Memory.

[XEQ] "EMDIR" MATRP P012 A D100 All these files have been read by READEM.
TEXTE A040

... INSTRUCTIONS FOR READEM ...

1) After storing the file name into the ALPHA register, [XEQ] "READEM" copies this file from the mass memory medium to X-Memory.

2) If there is no HP-IL module, NO HPIL is displayed.

3) If the file is not on the medium, FL NOT FOUND is displayed.

4) If there is not enough space in X-Memory, NO ROOM is displayed. In this case add one or two X-MEMORY modules.

5) If the HP-IL module is plugged in, but there is no mass memory device on the loop, "NO DRIVE" is displayed.

6) If the file specified was not created by [WRTEM], "FLTYPE ERR" is displayed.

NOTA: Before loading a set of files, [READEM] purges the X-Memory.

STACK:
The stack is unaffected by [READEM].

INVERSE FUNCTION: WRTEM.
[RG] is a function which makes easier the entry of functions beginnings with "RG". This function should be assigned to a key. For instance, assigne [RG] to the [LN] key.

ASN "RG" 15

Keystrokes : [ ] [ASN] [ALPHA] [R] [G] [ALPHA] [LN]. Put the calculator in USER mode. Now to execute or program a function beginning with "RG", for instance "RGVIEW", stroke the following keys :

[RG] (LN key) [ALPHA] [V] [I] [E] [W] [ALPHA]

This sequence is equivalent to :

[XEQ] [ALPHA] [R] [G] [V] [I] [E] [W] [ALPHA]

So you save 2 keystrokes, every time you use a function beginning with "RG".

INSTRUCTIONS FOR RG

1) Assign [RG] to a key and set USER mode.

2) To execute or input a function beginning with "RG" :

[RG] (Assigned previously)

[ALPHA]

...function name without the 1st two letters.

...(e.g. SUM for RGSUM).

[ALPHA]
- OPERATIONS BETWEEN REGISTERS -

- Addition or substraction of two vectors -

\[ \text{RG+/-} \]

\([\text{RG+/-}] \) (ReGisters + or -) adds or substracts, element by element, two vectors whose pointers are in registers \(X\) and \(Y\). The sign of the \(X\) register specifies the type of operation.

- Multiplication of two vectors, element by element -

\[ \text{RG*} \]

\([\text{RG*}] \) (ReGisters *) multiplies the two vectors, element by element, whose pointers are in registers \(X\) and \(Y\).

- Divide two vectors, element by element -

\[ \text{RG/} \]

\([\text{RG/}] \) (ReGisters /) divides, element by element, the two vectors, whose pointers are in the \(X\) and \(Y\) registers.

Example: in the Array below:

- replace the 1st column by the addition, element by element, of the 3rd and 1st column;
- then compute the square of the elements of 4th column;
- finally, divide each of those squares by the 4 first elements of the first line.

The array pointer is saved in register R00.

Array before execution:

**N.B.** In each box are shown the register number and its initial value.

<table>
<thead>
<tr>
<th></th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no 1</td>
<td>142</td>
<td>20</td>
<td>857</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line no 2</td>
<td>285</td>
<td>12</td>
<td>714</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line no 3</td>
<td>428</td>
<td>22</td>
<td>571</td>
<td>26</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line no 4</td>
<td>714</td>
<td>32</td>
<td>285</td>
<td>34</td>
<td>4</td>
</tr>
</tbody>
</table>

**ARRAY B**
Keystrokes : Display :

[CF] 28 [FIX] 5
[1] [RCL] 00 25.04405
[COLPT] 25.04005 First column pointer.
[3] [RCL] 00 25.04405
[COLPT] 27.04205 3rd column pointer.
[XEQ] "RG+-" 25.04005 Pointer of the output vector.

Now, you can check that registers R25, R30, R35, and R40, which make up the first column, are equal to 999.

[4] [RCL] 00 25.04405
[COLPT] [ENTER] 28.04305 X and Y contains the 4th column pointer.
[RG] *** 28.04305

Now, the elements of the 4th column are:

R28= 1600 R33= 196 R38= 576 R43= 1.156

[1] [RCL] 00 25.04405
[LINPT] 25.02900 First line pointer.
[XEQ] "RG/" 28.04305

At end, the 4th column contains the result of the division, and the array is the following.

<table>
<thead>
<tr>
<th>column no1 no2 no3 no4 no5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>line no1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>line no2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>line no3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>line no4</td>
</tr>
</tbody>
</table>

INSTRUCTIONS FOR RG+- RG* RG

1) Functions [RG+-], [RG*] and [RG/] require two pointers. The operand pointer must be in the Y-register and the operator pointer in the X-register.

2) The results are stored into the block pointed by the Y-register.

3) After execution, the X-register contains the resulting vector pointer.
STACK:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>Y: t</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: t</td>
</tr>
<tr>
<td>Y: pointer no1</td>
<td>Y: z</td>
</tr>
<tr>
<td>X: pointer no2</td>
<td>X: pointer no1</td>
</tr>
<tr>
<td>L: l</td>
<td>L: pointer no2</td>
</tr>
</tbody>
</table>
- SCALAR TO REGISTERS OPERATIONS -

- Add constant to registers - \[ \text{RG+Y} \]

\[ \text{RG+Y} \] (ReGisters + Y) adds the Y-register value to the registers specified by the X-register.

- Multiply registers by constant - \[ \text{RG*Y} \]

\[ \text{RG*Y} \] (ReGisters * Y) multiplies the registers specified by the X-register, by the Y-register value.

- Divide registers by constant - \[ \text{RG/Y} \]

\[ \text{RG/Y} \] (ReGisters / by Y) divides the registers specified by the X-register, by the Y-register value.

Example: In the Array B:

- Subtract 5 to the first column;
- multiply by 2 the 3rd line;
- divide by 6 the 5th column.

The array pointer is saved in register R00.

<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td>line no2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>line no3</td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>line no4</td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>

ARRAY B

Keystrokes : Display :

5 [CHS] [ENTER*] \(-5.00000\) Input the constant.
1 [RCL] 00 25.04405
[COLPT] 25.04005 First column pointer.
[RG] "+Y" 25.04405 Pointer of result vector.
You can check that R25, R30, R35, R40 contains respectively 4, 1, 6, and 11; it is the first column of the array.

2 [ENTER" ] 2.00000 Input the constant.
3 [RCL] 00 25.04405
[LINPT] 35.03900 3rd column pointer.
[RG] "*Y" 35.03900

Now the 3rd line values have been multiplied by 2. R35= 12, R36= 24, R37= 26, R38= 28, R39= 30.

6 [ENTER" ] 6.00000 Input constant.
5 [RCL] 00 25.04405
[COLPT] 29.04405 5th column pointer.
[RG] "/Y" 29.04405

After all these transformations, array B is:

<table>
<thead>
<tr>
<th>line no1</th>
<th>line no2</th>
<th>line no3</th>
<th>line no4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>R29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>R34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>26</td>
<td>28</td>
</tr>
<tr>
<td>R39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>R44</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ARRAY B

INSTRUCTIONS FOR RG+Y RG*Y RG/Y

1) [RG+Y], [RG*Y], [RG/Y], need a constant in the Y-register, and a pointer in the X-register.

2) Operations are directly performed on the register value, so results replace initial values.

STACK:

Input:

T: t
Z: z
Y: scalar
X: pointer
L: l

Output:

T: t
Z: z
Y: scalar
X: pointer
L: l

The stack is unchanged by [RG+Y], [RG*Y], [RG/Y].
- Registers to ALPHA or ALPHA to registers -

[RGAX] (ReGisters—Alpha by X) performs two functions:

1) If X<0, it copies the ALPHA register to the registers specified by the register pointer in the X-register;

2) If X>=0, it appends to the ALPHA register the contents of the registers specified by the register pointer in the X-register.

Example: The string "ABCDEFGHIJKLMNOPQRSTUVWX" is in the ALPHA register. To save it in even registers, starting from R10, use the following sequence:

Keystrokes : Display :
10.00002 [CHS] -10.00002 Pointer. The negative value indicates that it stores ALPHA to registers.
[RG] "AX" -17.00002 The pointer specified the register following the last one used by [RGAX].
[RCL] 10 ABCDEF first 6 characters.
[RCL] 12 GHIJKL next 6 characters.
[RCL] 14 MNOPQR next 6 characters.
[RCL] 16 STUVWX last 6 characters.

Now, if you want to retrieve registers R12 and R16 to the ALPHA register:

Keystrokes : Display :
12.00004 12.00004 Register pointer to recall the string.
[XEQ] "CLA" 12.00004 clear the ALPHA register.
[RG] "AX" 17.00004 Indicate next register.
[ALPHA] GHIJKLSTUVWX Recall ends, when the last character of the string is found.

INSTRUCTIONS FOR RGAX

1) The [RGAX] function can be used to save all the ALPHA register to the registers specified by the register pointer in the X-register. In this case the pointer must be negative. When the calculator saves a string, it adds an 'End-of-String' indicator to the last register used. This indicator is used when the string is recalled; it is invisible, but a modification of the last register destroys the indicator.

2) The [RGAX] function can also be used to recall a string that was previously saved in a set of registers. In this case, the pointer should be positive. The string is appended to the ALPHA register string. If the new string is more than 24 characters long, only the last 24 ones remains in the ALPHA register. The leftmost characters are lost. Loading stops when an 'End-of-String' indicator is recalled, or if there is no indicator, when a numeric value is found. In this case, the numeric value is appended, in the current format, to the ALPHA register, as it would be with [ARCL].

3) In both cases, [RGAX] saves the initial pointer in LASTX, and leaves a ± bbb, eeeii pointer in the X-register. bbb is the last register used +1, eeeii is the eeeii part of the initial pointer. The first three decimal digits of the initial pointer can be anything, because [RGAX] does not use them.
STACK:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: t</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: z</td>
</tr>
<tr>
<td>Y: y</td>
<td>Y: y</td>
</tr>
<tr>
<td>X: Initial pointer</td>
<td>X: New pointer</td>
</tr>
<tr>
<td>L: l</td>
<td>L: Initial pointer</td>
</tr>
</tbody>
</table>
- Copy or exchange registers -

[RGCOPY] (ReGisters COPY) performs two types of operations:

If \( X \geq 0 \), \( \text{[RGCOPY]} \) copies the registers specified by the \( X \)-register pointer, to the registers specified by the \( Y \)-register pointer.

If \( X < 0 \), \( \text{[RGCOPY]} \) swaps the registers specified in the \( X \)-register, with the ones specified in the \( Y \)-register.

Example: In the array \( B \), copy the first column to the 3rd one, then swap the 1st and 2nd columns.

<table>
<thead>
<tr>
<th>column no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
</tr>
<tr>
<td>line no2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>line no3</td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
</tr>
<tr>
<td>line no4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
</tbody>
</table>

ARRAY \( B \)

Suppose that the array pointer is in register \( \text{R00} \).

Keystrokes: Display:

3 \( \text{[RCL]} \) 00 \( \text{[COLPT]} \) 27.04205 Destination pointer.
1 \( \text{[LAST X]} \) \( \text{[COLPT]} \) 25.04005 Origin pointer.
\( \text{[XEQ]} \) "RGCOPY" 27.04205 Destination pointer.
\( \text{[RgVIEW]} \) list the 3rd column \( R27=1, \ldots, R42=16 \).
1 \( \text{[RCL]} \) 00 \( \text{[LINPT]} \) 25.02900 1st Pointer.
2 \( \text{[LAST X]} \) \( \text{[COLPT]} \) \( \text{[CHS]} \) -26.04105 2nd pointer.
\( \text{[Rg]} \) "COPY" 25.02900 The stack moved down.
The final array is:

<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no1</td>
<td>2</td>
<td>7</td>
<td>12</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>line no2</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>line no3</td>
<td>11</td>
<td>1</td>
<td>11</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>line no4</td>
<td>16</td>
<td>4</td>
<td>16</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>

**ARRAY B**

**INSTRUCTIONS FOR RGCOPY**

1) The sign of the pointer in the X-register specifies if the registers have to be copied (X>=0) or swapped (X<0).

2) Copy is performed from the registers specified by the pointer in the X-register to the registers specified by the pointer in the Y-register. At the end the stack moves down.

3) Swap is done between the registers specified in the X and Y registers. If there is no overlapping, swap begins with the lower register number. If there is overlapping, the calculator begins, one way or another (i.e. upwards or downwards), so that no information is lost.

**STACK:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>T: t</td>
<td>T: t</td>
</tr>
<tr>
<td>Z: z</td>
<td>Z: t</td>
</tr>
<tr>
<td>Y: Destination pointer</td>
<td>Y: z</td>
</tr>
<tr>
<td>X: Origin pointer</td>
<td>X: Destination pointer</td>
</tr>
<tr>
<td>L: l</td>
<td>L: Origin pointer</td>
</tr>
</tbody>
</table>
Initialization of registers by X -

**[RGINIT]** (ReGisters INITialize) performs two kinds of initializations:

If \( X \geq 0 \) **[RGINIT]** stores zero in all the registers specified by the pointer in the \( X \)-register.

If \( X < 0 \) **[RGINIT]** stores integers, from 1 to \( N \), into the registers specified by the pointer in the \( X \)-register, \( N \) being the numbers of registers specified.

**Example**: In the array B, which pointer is saved in register \( R00 \), columns 3 and 5 will be zeroed, then the columns will be numbered from 1 to 5, through the first line.

<table>
<thead>
<tr>
<th>Column no</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Line 1</strong></td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td><strong>Line 2</strong></td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td><strong>Line 3</strong></td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td><strong>Line 4</strong></td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
</tbody>
</table>

Keystrokes:  
3 \([\text{RCL}]\) 00 \([\text{COLPT}]\)  
\(\times \text{EQ} \) "RGINIT"  
5 \([\text{LASTx}]\) \([\text{COLPT}]\)  
\(\times \text{EQ} \) "RGINIT"  
1 \([\text{LASTx}]\) \([\text{LINPT}]\) \([\text{CHS}]\)  
\(\times \text{EQ} \) "RGINIT"

Display:  
27.04205 3rd column pointer.  
27.04205 Initialize 3rd column to zero.  
29.04405 5th column pointer.  
29.04405 Initialize 5th column to zero.  
-25.02900 Negative sign to indicate an  
-25.02900 Initialization with integers 1 to \( N \).

<table>
<thead>
<tr>
<th>Column no</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Line 1</strong></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td><strong>Line 2</strong></td>
<td>F</td>
<td>G</td>
<td>0</td>
<td>I</td>
<td>0</td>
</tr>
<tr>
<td><strong>Line 3</strong></td>
<td>K</td>
<td>L</td>
<td>0</td>
<td>N</td>
<td>0</td>
</tr>
<tr>
<td><strong>Line 4</strong></td>
<td>P</td>
<td>Q</td>
<td>0</td>
<td>S</td>
<td>0</td>
</tr>
</tbody>
</table>
INSTRUCTIONS FOR RGINIT

1) When the register pointer, in the X-register, is positive, the specified registers are initialized to zero.

2) When the register pointer in the X-register is negative, the specified registers are initialized with integers, starting with 1 and incrementing it by 1 for each register, up to the last one.

STACK:

The stack is unchanged by the execution of [RGINIT].
Number of registers specified by a pointer

[RGNb] (ReGisters, NumBer of) returns the number of registers specified by the pointer in the X-register.

Example: Compute the number of elements of an array, whose pointer is saved in register R00; then compute the number of registers per line.

Keystrokes: Display:

[XEQ] "RGNb" 20.00000 The array contains 20 registers.
[XEQ] "RGNb" 5.00000 There are 5 registers per line.

INSTRUCTIONS FOR RGNb

[RGNb] returns to the X-register, the number of registers specified by a bbb.eeeii pointer to the X-register. The pointer is saved in LASTX.

STACK

Input: Output:

T: t T: t
Z: z Z: z
Y: y Y: y
X: Pointer X: Number of elements
L: l L: Pointer
- Sum of registers -

[RGSUM] (ReGisters, SUM of) returns to the X-register the sum of the registers specified by the pointer in the X-register.
If the pointer is negative, [RGSUM] performs the sum of the absolute values of the specified registers.

Example: In the array F, whose pointer is saved in register R00, one wants the total of the 1st column, and the sum of the 4th column, but considering the absolute value of the elements.

<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no1</td>
<td>-14</td>
<td>15</td>
<td>21</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>line no2</td>
<td>7</td>
<td>13</td>
<td>19</td>
<td>-20</td>
<td>1</td>
</tr>
<tr>
<td>line no3</td>
<td>0</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>line no4</td>
<td>23</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>17</td>
</tr>
<tr>
<td>line no5</td>
<td>16</td>
<td>22</td>
<td>3</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Keystrokes: Display:

1 [RCL] 00 [COLPT] 25.04505 1st column pointer.
[RG] "SUM" 32.00000 Sum of elements.
[XEQ] "RGSUM" 60.00000 Sum of absolute values.

INSTRUCTIONS FOR RGSUM

[RGSUM] returns to the X-register, the sum of the registers specified in the X-register.
If the pointer in the X-register is negative, [RGSUM] performs the sum of the absolute values of the registers.

STACK:

Input:       Output:
T:  t         T:  t
Z:  z         Z:  z
Y:  y         Y:  y
X:  Pointer   X:  Sum
L:  l         L:  Pointer
APPLICATION PROGRAMS FOR RGSUM

1) In the array F, we want to put in the 3rd column, the percentage of the values of the 4th column, related to their sum.

Keystrokes :: Display :

3 [RCL] 00 [COLPT] 27.04705 Destination pointer.
2 [LAST X] [COLPT] 26.04605 Origin pointer.
[RGCOPY] 27.04705 Copy the 2nd column to the 3rd one.
[XEQ] "RGSUM" 60.00000 Sum of elements.
[LAST X] [X<>Y] 60.00000 Save the pointer.
100 [/] 0.60000 To compute the percentage.
[X<>Y] [RG/Y] 27.04705 End.

Now array F is:

```
column  no1  no2  no3  no4  no5
  line no1    R25 | R26 | R27 | R28 | R29 |
                   | 14  | 15  | 25  | 2   | 8   |
  line no2    R30 | R31 | R32 | R33 | R34 |
                   | 7   | 13  | 21.6| -20 | 1   |
  line no3    R35 | R36 | R37 | R38 | R39 |
                   | 0   | 6   | 10  | 18  | 24  |
  line no4    R40 | R41 | R42 | R43 | R44 |
                   | 23  | 4   | 6.6 | 11  | 17  |
  line no5    R45 | R46 | R47 | R48 | R49 |
                   | 16  | 22  | 36.6| 9   | 10  |
```

ARRAY F

The 3rd column now holds the percentages!
[RGVIEW] (ReGisters VIEW) is a multi-mode function to view, and/or edit registers.

**Example**: the following sequence performs several viewing of the array I. In some cases, registers are modified.

<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no1</td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td>line no2</td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td>line no3</td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td>line no4</td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
</tbody>
</table>

**Keystrokes**: 
- **Display**: 

- **[CF]** 28 [**[FIX]**] 6 [**[<-]**] 0.000000
- **[RCL]** 00 [**[RGVIEW]**] 25= 1.000000
- 30= 6.000000 View the first column.
- **[R/S]** 35= 11.00000 Halt the catalog.
- **[SST]** 40= 16.00000 Single stepping is
- **[BST]** 35= 11.00000 Possible in both direction.
- **[<-]** 25.044050 Exit the catalog.
- **[CLINC]** 25.044000 Register pointer.
- **[RGVIEW]** 25= 1.000000
- 26= 2.000000 Automatic stepping in the
- 27= 3.000000 registers and visualization
- 28= 4.000000 of them.
- **[ON]** Turn the calculator off, then back on.
- **[ON]** [**[CHS]**] -25.044000 Pointer to stop at the first value.
- **[RGVIEW]** 25= 1.000000
- 15 25= 15_ Input directly
- **[CHS]** 25= -15_ Into the register,
- **[EEX]** 25= -15_ exactly as normal keyboard
- **2 [**[CHS]**] 25= 15 -2_ input.
- **[R/S]** 26= 2.000000 Validate data.
- **[BST]** 25= -0.15000 Verification.
- **[SST]** [**[ALPHA]**] 26= 2.000000 Set ALPHA mode.
- **ABCDEF** 26= ABCDEF_ Up to 6 characters
are allowed.
Corrections can be made.
Validation and verification.
The ALPHA mode is still on.
Numeric mode.
Unchanged: no validation with [R/S].
Exit.

Array pointer.
Array name.
Only the last character is used as
the array name.
stepping in the array,
is automatic.
(R/S) halts it.
The element coordinates
are displayed on the left.
Quick and clear array
input is possible.

Vector pointer.
1st element of the 1st column.
2nd element (R30).
3rd column pointer.
1st element (R27).

Input the column,

End input and clear ALPHA mode
Creation of a new
pointer.

In this mode, [RGVIEW]
allows inputs while the former
value or string
is still displayed.
In this catalog mode,
zeroes are ignored.
INSTRUCTIONS FOR RGVIEW

1) [RGVIEW] is a general purpose display, input, and print function, for main memory registers.

2) The X-register pointer specifies registers and [RGVIEW] mode. It is a bbb.eeeiiij pointer.

If X>0 : View in sequence the registers specified, up to the end of the specified block, or up to an interruption with the [R/S] key.

If X<0 : View and stop on the 1st register specified. Use [SST] to skip to the next register. The [R/S] key resumes the listing.

When j is an odd number, registers equal to 0 are ignored (skipped).

If j= 0 or 1, it is a standard catalog: the register number and its value are displayed.

If j= 2 or 3, [RGVIEW] displays the array elements to the contents of the array name and elements coordinates to the left.

If j= 4 or 5 [RGVIEW] displays the register value, followed by "=". Input is performed with the old value still in the display.

Example: Display LUNDI=
Input LUNDI= 10

If j= 6 or 7 [RGVIEW] displays the vector name (1 dimension), the element coordinate (index), and its value.

In ALPHA mode, only the last six characters input are retained for entry.

A printer in NORMal or TRACE mode prints the register catalog output by [RGVIEW].

3) [RGVIEW] works like a CATalog ([SST] and [BST] are allowed).

STACK:

Input: Output:

T: t T: t
Z: z Z: z
Y: y Y: y
X: pointer X: pointer
L: l L: previous pointer
[SORT] sorts the registers specified in the X-register.

**Example:** In the array A below:

1) Sort in increasing order the values in the 2nd col.

2) Sort in decreasing order the values in the 3rd col.

```plaintext
<table>
<thead>
<tr>
<th>column no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td>line no1</td>
<td>14</td>
<td>B</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>line no2</td>
<td>7</td>
<td>13</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>line no3</td>
<td>0</td>
<td>A</td>
<td>-12</td>
<td>18</td>
</tr>
<tr>
<td>line no4</td>
<td>23</td>
<td>99</td>
<td>50</td>
<td>11</td>
</tr>
</tbody>
</table>

Keystrokes: Display:

[XEQ] "SORT" SORTING Sorting in progress ...
26.04105 Done.

3 [LASTx] [COLPT] [CHS] -27.04205 3rd column pointer; the negative pointer indicates a decreasing order.
[XEQ] "SORT" SORTING Sorting in progress ...
-27.04205 Done.

<table>
<thead>
<tr>
<th>column no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td>line no1</td>
<td>14</td>
<td>13</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>line no2</td>
<td>7</td>
<td>99</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>line no3</td>
<td>0</td>
<td>A</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>line no4</td>
<td>23</td>
<td>B</td>
<td>-12</td>
<td>11</td>
</tr>
</tbody>
</table>

ARRAY A
```
INSTRUCTIONS FOR SORT

1) [SORT] sorts either numeric values, or alpha strings. Strings are sorted according to their ASCII code and they are considered as being greater numeric values.

2) The pointer in the X-register specifies the registers to sort.

3) If X>=0, registers are sorted in increasing order.

4) If X<0, registers are sorted in decreasing order.

5) While sorting is in progress, if there is no message yet in the display, "SORTING" is displayed.

STACK:

[SORT] leaves the stack unchanged.
[STO>L] (STOrE by L ) stores the X-register value, into the register specified by the integer part of the L-register pointer. It also increments this pointer; stack lift is disabled.

Example: To input all the elements of the 1st line, of a 4 lines, 5 columns array, beginning with register R25.

Keystrokes:

1 1_
[RCL] 00 25.04405 Recall the pointer.
[LINPT] 25.02900 Compute 1st line pointer.
50 50_ 1st line, 1st element.
[XEQ] "STO>L" 50.00000 Store it in R25.
[VIEW] [. ] [L] 26.02900 Pointer has been incremented.
60 60_ 2nd element.
[XEQ] "STO>L" 60.00000 Store 2nd element.
70 70_ 
[XEQ] "STO>L" 70.00000 
80 80_ 
[XEQ] "STO>L" 80.00000 
90 90_ 
[XEQ] "STO>L" 90.00000 
[LASTX] 30.0290

INSTRUCTIONS FOR STO>L

[STO>L] uses the L-register, as an address pointer to store the X-register value.

[STO>L] transfers the X-register value, to the register specified by the L-register. Stack lift is not done, so several values can be input, without altering registers Y, Z, T. Furthermore, the pointer in the L-register is automatically incremented, resulting in significant code savings within a program.

STACK:

Input: Output:

T: t T: t
Z: z Z: z
Y: y Y: y
X: value to store X: value stored
L: bbb L: bbb+1

Note: The decimal part of the L-register is ignored.

NOTA: If there is an alpha string in the L-register, ALPHA DATA is displayed.
APPLICATION PROGRAM FOR STO>L

1) [STO>L] was designed, to input register values programmatically. In order to input, the 1st colonne of the array B, whose pointer is in register R00, use the following sequence:

1 RCL 00 COLPT STO L 50 STO>L 60 STO>L 70 STO>L 80 STO>L

<table>
<thead>
<tr>
<th>column</th>
<th>no1</th>
<th>no2</th>
<th>no3</th>
<th>no4</th>
<th>no5</th>
</tr>
</thead>
<tbody>
<tr>
<td>line no 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R25</td>
<td>R26</td>
<td>R27</td>
<td>R28</td>
<td>R29</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R30</td>
<td>R31</td>
<td>R32</td>
<td>R33</td>
<td>R34</td>
</tr>
<tr>
<td>line no 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R35</td>
<td>R36</td>
<td>R37</td>
<td>R38</td>
<td>R39</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R40</td>
<td>R41</td>
<td>R42</td>
<td>R43</td>
<td>R44</td>
</tr>
<tr>
<td>line no 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line no 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ARRAY A
- Extraction or justification of a substring -

[SUB$] (SUB string) extracts a substring from the ALPHA register, or performs a right or left justification of a string, adding spaces to it.

Example: To extract 7 characters, starting with "C", from the string 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', which is in the ALPHA register:

Keystrokes: | Display:
---|---
2.08 | 2.08
[XEQ] "SUB$" | 2.0800
[ALPHA] | CDEFGHI

To right-justify in a 10 character field:

[ALPHA]
10
[CHS]
[XEQ] "SUB$" | -10.0000
[ALPHA] | CDEFGHI

The string in ALPHA register is 10 characters long.

To put 5 spaces at the right of the string:

[ALPHA]
15 | -10.0000
[XEQ] "SUB$" | 15.0000
[ALPHA] | CDEFGHI
[APPEND] | DEFGHI

INSTRUCTIONS FOR SUB$

[SUB$] modifies the ALPHA register as specified by the X-register.

- If the X-register number is an integer, the calculator extracts the \(|X|\) left most characters of the initial string. If the initial string has less than \(|X|\) characters, spaces are added to get a \(|X|\) character string; spaces are added to the left if X<0, to the right if X>0.

- If the number in the X-register has a decimal part (bb,ee), the calculator extracts the substring comprising of the characters from the position bb to the position ee. (The first character is at position 0). If ee is higher than the position of the last character of the string, the substring is the initial string, with spaces added to get a ee-bb+1 character string. Spaces are added to the left if X<0, to the right if X>0. Beware that the sign of the X-register is ignored if ee is not higher than the position of the last character in the ALPHA register.

If bb is higher than the position of the last character of the initial string, [SUB$] puts a string of ee-bb+1 space characters in the ALPHA register.
STACK:

The stack is not changed by [SUB$].

NOTA: if the ALPHA register string is 24 character long, the calculator puts in the front of the string, characters with code 0, which are displayed as small dashes before the string.
[TF55] (Toggle Flag 55) toggles flag 55, which indicates that a printer is connected to the HP41. This flag cannot be modified by the user without the PANAME ROM. The TF55 function:

1) Sets flag 55 when there is no printer attached to the HP41; this eases the use of some programs (for instance in Application Pacs), which must be executed with flag 21 set (Printing enabled). So you can use them as subprograms, because when flag 55 is cleared, those programs halts at every VIEW or AVIEW instructions. With [TF55] there is no more interruption.

2) Clears flag 55, when a printer is attached to the HP-41: this speed up programs when the printer is not used. Another [TF55] puts the printer back to use.

INSTRUCTIONS FOR TF55

1) To set flag 55, when it is cleared, execute [TF55].

2) To clear flag 55, when it is set, execute [TF55].
[VKEYS] (View KEYS) lists the key assignments to the HP 41 built-in display. For instance, if the PROMPT function is assigned to the "ENG" key (shifted [3], key code -74), the calculator will display:

-74 PROMPT

Key listing can be:

- Suspended by pressing and holding down any key but [R/S] or [ON];

- Terminated with the [ON] or [R/S] key. [ON] also turns off the calculator off.

Nota: [VKEYS] is not programmable.
[WRTEM] (WRiTe Extended Memory) copies all the extended memory to a mass medium (HP82161A Tape drive or HP9114 Disk Drive).

Example:

Keystrokes: Display:
[XEQ] "EMDIR" MATRP P012
A D100 Samples contents of
TEXTE A040 Extended Memory.

[ALPHA] "MAT 3" [ALPHA] 600.00 Put the file name in ALPHA
[XEQ] "WRTEM" 600.00 All the X-Memory files have been written to the mass medium.

INSTRUCTION FOR WRTEM

1) Put in the ALPHA register the filename in which all the X-Memory should be saved; then [XEQ] "WRTEM" copie all the X-Memory to this file.

2) If there is no HP-IL ROM, NO HPIL is displayed.

3) If a file exists with the same name on the mass medium, it is replaced by the new one.

STACK:

The stack is unchanged by [WRTEM].

INVERSE FUNCTION: READEM.
The X<>F function swaps the X-register and a imaginary register F, which represents the status of flags 0-7. This representation is an integer, in the range 0-255, which is the sum of the values related to the flags set:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
</tr>
</tbody>
</table>

For instance, if flags 0, 1, and 3 are set, and flags 2, 4, 5, 6, and 7 are cleared, the "F" register value is:

1 (value of flag 0)  
+2 (value of flag 1)  
+8 (value of flag 3)  
=11

INSTRUCTIONS FOR X<>F

To swap the current status of flags 0-7, with a new one:

1) Compute (see above), the number related to the new status of flags 0-7 and put it in the X-register;

2) Execute X<>F.

Now, the X-register number contains the old status of flags 0-7, and flags 0-7 are set to the new status.

APPLICATION PROGRAM FOR X<>F

The XFLAGS program gives you up to 80 new flags. Usage of these extended-flags (0-79) is as follows:

- Set the Nth X-flag : Put N in the X-register and XSF.
- Clear the Nth X-flag : Put N in the X-register and execute XCF.
- Test the Nth X-flag : Put N in the X-register and execute XFS?.
  After XFS?, flag 08 reflects the status of the X-flag being tested.

XSF, XCF and XFS? programs uses the stack and registers R00 to R09. XFS? also uses flag 08.
**XFLAGS** program listing:

```
LBL "XFLAGS" .009 RGINIT RDN RTN
LBL "XSF" XEQ 00 SF IND Y GTO 01
LBL "XCF" XEQ 00 CF IND Y GTO 01
LBL "XFS?" XEQ 00 CF 08 FS? IND Y SF 08
LBL 01 X<>F STO IND Z R^ RTN
LBL 00 STO Y 8 /MOD RCL IND Y X<>F .END.
```

Note: XEQ "XFLAGS" clears all X-flags 00 thru 79.
- Compare X and a register -

**X...NN?**

Functions `X#NN?`, `X<=NN?`, `X<NN?`, `X<=NN?`, `X>=NN?` and `X>NN?` work as the built-in test functions `(X=Y?)` of the HP-41, but they compare the X-register contents with the contents of any register specified in the Y-register.

Furthermore these functions also compare alphanumeric strings.

**INSTRUCTIONS FOR X...NN?**

To compare the contents of the X-register with that of data register r:

<table>
<thead>
<tr>
<th>If the r register is</th>
<th>Put in Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>- a data register Rnnn</td>
<td>- the number nnn</td>
</tr>
<tr>
<td>- the Z-register</td>
<td>- the string 'Z'</td>
</tr>
<tr>
<td>- the T-register</td>
<td>- the string 'T'</td>
</tr>
<tr>
<td>- the L-register</td>
<td>- the string 'L'</td>
</tr>
</tbody>
</table>

In RUN mode, the calculator displays YES or NO according to the result of the test.

In a program, X...NN? behaves like any built-in test function. The line following the test is executed if the test is true, or it is skipped if the test is false.

These functions compare numeric and alphanumeric functions, according to the following assumptions:

1) A number is always less than a string.

2) Strings are compared according to their character codes. (e.g.: 'AB0' < 'ABA' because the code of '0' is 48 and that of 'A' is 65.

3) A short string identical to the beginning of a larger one is considered as being less than the larger one (e.g. "ABC" < "ABCD").
[Y/N] is useful in programs, which ask for an answer Yes or No.

Example: The following sequence displays the question:

EXIT Y/N?

If the user answers Yes (Y key), the program resumes execution at label 00, if the user answer NO (N key) the program resumes execution at label 01:

"EXIT" 1,000
Y/N
GTO 00
GTO 01

INSTRUCTIONS FOR Y/N

The Y/N function can only be used in a program.

1a) To ask a question in the form:

message Y/N?

Put the message (max. 7 characters) in the ALPHA register and execute Y/N;

1b) To ask another type of question put the message in the ALPHA register, execute AVIEW then Y/N (one that does not contain Y/N ? explicitly).

2) In both cases, when the Y/N function is executed, the calculator suspends program execution and waits for a keystroke:

- If the key pressed is ON, the calculator is turned off:
- If the key pressed is R/S, program execution is stopped and the program pointer is set to the line following Y/N;
- If the key pressed is Y (Yes) or O (Oui), program execution resumes at the line following immediately Y/N;
- If the key pressed is N (No), the line following Y/N is skipped, and the program resumes execution at the second line after Y/N (As for a false test; see the description of the X=Y? function in the HP41 manual).
- Any other key is ignored.
Appendix ON

With the PANAME ROM plugged in, MEMORY LOST is not the only ON/Key combination that is meaningful to the HP 41: 6 new functions are possible when you switch on the HP-41.

Notation: ON/+ represents the function performed when you turn the calculator on with [ON], while the + key is held down. Also you must release [ON] before +.
For instance, with ON/↔ you get a MEMORY LOST.

ON/. Change the numeric display format: from the "American" one (1,234.25), to the "European" one (1.234,25). This function is built in the HP-1x calculators.
In fact ON/. toggles flag 28.

ON/K Clear all user key assignations.

ON/A Set the "A key assignment set", to the top rows. If one key was already assigned to some function or program, its key assignment is not modified.

<table>
<thead>
<tr>
<th>ATOXL</th>
<th>ALENG</th>
<th>ATOXX</th>
<th>ANUMDEL</th>
<th>ATOXR</th>
<th>Shifted 1st row</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTOAL</td>
<td>AROT</td>
<td>YTOAX</td>
<td>ANUM</td>
<td>XTOAR</td>
<td>Unshifted 1st row</td>
</tr>
</tbody>
</table>

ON/M Like ON/A, but with the following "H key assignment set".

<table>
<thead>
<tr>
<th>STO-L</th>
<th>BRKPT</th>
<th>COLPT</th>
<th>AD-LC</th>
<th>RGVIEW</th>
<th>Shifted 1st row</th>
</tr>
</thead>
<tbody>
<tr>
<td>RG</td>
<td>BLDPY</td>
<td>LINPT</td>
<td>LC-AD</td>
<td>CLINC</td>
<td>Unshifted 1st row</td>
</tr>
</tbody>
</table>
ON/T

Like ON/A, but with the following "T key assignment set".

<table>
<thead>
<tr>
<th>AXIS</th>
<th>BOX</th>
<th>SETORG</th>
<th>RMOVE</th>
<th>*CSIZE</th>
<th>Shifted 1st row</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unshifted 1st row</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Shifted 2nd row</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unshifted 2nd row</td>
</tr>
</tbody>
</table>

ON/V

Like ON/A, but with the following "V key assignment set".

<table>
<thead>
<tr>
<th></th>
<th>SCRLUP</th>
<th>CLEAR</th>
<th>XYTAB</th>
<th>CTYPE</th>
<th>Shifted 1st row</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td></td>
<td>CLEARO</td>
<td>CSRL</td>
<td>CSRR</td>
<td>Unshifted 1st row</td>
</tr>
<tr>
<td></td>
<td>SCRLX</td>
<td>CSRVX</td>
<td>CSRUP</td>
<td>CSROFF</td>
<td>Shifted 2nd row</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unshifted 2nd row</td>
</tr>
</tbody>
</table>
The HP-41 uses the "Reverse Polish Notation" (RPN), to solve complex problems, without requiring parentheses and with a minimum of keystrokes. This system was created by a famous polish mathematician, Lukasiewicz, and not by the Hewlett-Packard company.

- A TIME SAVING SYSTEM: the access to memory registers is quicker in the stack with any memory partition, than on any other type of calculator;

- A SPACE SAVING SYSTEM: an intermediat result, which does not use a register, let it free for something else;

- Furthermore the STACK avoid a COMPLEX MEMORY MANAGEMENT: one can use a subprogram without modifying it, because of the memory implantation of the variables in the calling program.

So, subprograms parameters are passed through the stack, calculations are done in the stack, and results are returned to the stack: a subprogram is general, and easy to use.

In general any arithmetic processing, with up to 4 values, can be performed in the stack.

**Example 1**: roots of a \( ax^2 + bx + c = 0 \) equation.

To use the program given below:

\[
\begin{align*}
c \text{ ENTER}^* b \text{ ENTER}^* a \text{ XEQ "ROOTS" .} \\
\text{LBL "ROOTS" ST/ Z CHS ST+ X / ENTER}^* X^2 \\
\text{RCL Z - SQRT RCL Y SIGN * + ST/ Y .END.}
\end{align*}
\]

For instance to solve the following equation set:

\[
x^2 + x -6 = 0 \text{ and } \\
3x^2 + 2x -1 =0 -6
\]

ENTER\(^* 1 \) ENTER\(^* \) XEQ "ROOTS" . \\
x\(^*\) = -3 and RDN x\(^*\) = 2

For the second one -1 ENTER\(^*\) 2 ENTER\(^*\) 3 XEQ "ROOTS" \\
x\(^*\) = -1 and RDN x\(^*\) = 0,3333

This example illustrates the easiness given by the arithmetic done directly in the stack.

**Example 2**: GCD (Greatest Common Divisor) of 2 numbers.

\[
\begin{align*}
01 \text{ LBL "GCD" LBL 02 MOD LASTX X<>Y X#0? GTO 02 + .END.} \\
91 \text{ ENTER}^* 65 \text{ XEQ "GCD" . X = 13,00}
\end{align*}
\]

In this example, you must put the two numbers in the X and Y registers, and the result is returned to the X-register.
Example 3: reduced fraction.

Generally you compute the GCD to get the reduced form of a fraction. Also with the GCD of a fraction it is easy to get the LCM (Least Common Multiple).

01 LBL "RF" STO Z X<>Y STO T XEQ "GCD" ST/ Z ST/ Y RDN ST* Z .END.

Application: 91 ENTER 65 XEQ "RF" returns X = 5 and Y = 7 so 91/65 = 7/5; the GCD and the LCM of 91 and 65 are Z = 455 and T = 13

Example 4: calculation with two fractions.

LBL "F/" X<>Y LBL "F*" ST* Z RDN ST* Z RDN GTO "RF" 09 LBL "F-" CHS LBL "F+" ST* T X<> Z ST* Z * RCL Z + X<>Y GTO "RF" .END.

Application: what resistance should be put in parallel with a 100 ohms one to get a result of 80 ohms?

1 ENTER^ 80 ENTER^ 1 ENTER^ 100 XEQ "F-" 1/X . So it is a 400 ohms one.

These examples illustrate, quite well, the powerful capabilities of the stack.