PANAME ROM

Functions and Array Handling Routines for the HP-41

Reference Handbook

PANAME ROM

PREFACE

When the HP-41 was introduced in 1979, it represented a significant evolution in the area of handheld programmable calculators; the ability to input, process and display alphanumeric characters. A real dialog was now possible between the user and the calculator. This new flexibility was also enhanced with the addition of audible tones.

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 "specialized machine language" is not adequate for the HP-41, "specialized assembly language" is better. In fact, for this calculator, HEWLETT-PACKARD has developed an advanced language, near FORTH, which places the HP-41 in a class by itself.

The comprehension of this potentially 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 (compiled) into a succession of micro-instructions, which are understandable to the HP-41 chip. This deciphering operation is the interpreting.

A SYMBOLIC LANGUAGE FOR THE HP-41.

A computer spends most of its time searching in its memory to transfer information 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 information:

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

At the machine language level this information is all numerical. But the average 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 that are analogous to a directory, where a telephone number is found using a surname. The advancement of a language can be measured 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. Every one of them can be created, modified, erased, and (if you own a mass storage device) saved or loaded independently.

Opposite this physical independence is a logical dependence. Any program can call a group of instructions belonging to another program as a subroutine. This sequence of instructions need only begin with an alphabetical label: LBL "X..." and end with RTN or END.

Therefore it is possible to divide a complicated program, into a sequence of easier ones (and so on) according to the valuable principles of ascending programming. The problems of data handling are left to lower level 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 designing the different stages of a program. If an error occurs, it is easy to correct because the error 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 may wish to have an assembly language version of them. This module is a good illustration of that. Most functions included in the PANAME ROM, were first created as subroutines in user language and published in 1982. The conception of the array handling techniques and of most of the functions are from this period.

PROGRAMMING THE HP-41.

The HP-41 has three programming levels.

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

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

They are general enough to be used several times in a program, even in different programs. They are liable to stay permanently in memory. A standardization of programming techniques saves time and effort. If a routine has all these conditions it can be considered as a new function for the HP-41 language: an illustration of the last quality of this language: Its capacity to evolve.

- Assembly language functions. They are the elements of the language itself. A function should be general, even more so 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.

One should use this module's functions to realize what simplifications they provide when dealing with peripherals. 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 as reading "SIN" instead of "31 04". "CLEAR" is better than "27 ACCHR 69 ACCHR OUTA": 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, everyone will see how the utilization of the stack, that we have recommended, prepared us for these new functions. The rise of the 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 those, who develop programs on the HP-41, will realize the great interest they have in acquiring the PANAME ROM.

PHILIPPE DESCAMPS

WARNING

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

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

For the moment the PANAME ROM as a user manual, which has been made as clear and precise as possible. But we realize 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 the author's 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.

FUNCTION OVERVIEW*

AID:
Returns the Accessory ID of the primary device to the X register.

ID:
Returns the device ID of the primary device to the ALPHA register.

FINDAID:
FIND an AID (or class) device specified by X (< 0 for a class) and return its address in X.

OUTAX:
OUTA with repetition. |X| indicates the number of repetitions

OUTCR:
Send decimal code 13 to the main device (Carriage Return)

OUTLF:
Send decimal code 10 to the main device (Line Feed)

OUTLFX:
Send decimal code 10 to the main device (Line Feed), |X| indicates the number of times to send the decimal code.

OUTSPX:
Send decimal code 32 to the main device (Space), |X| indicates the number of times to send the decimal code.

OUTXB:
Send the decimal code defined by the |X| register as a character Byte to the main device.

OUTYBX:
Send the decimal code defined by the |Y| register as a character Byte to the main device |X| times.

OUTa:
Similar to OUTA, but sets bit 7 with all the character bytes sent (for example for the inverse video on HP82163).

OUTaX:
OUTa repeated |X| times.

RCLSEL:
Returns the address of the main device in the |X| register. If the SELECT are > devices, RCLSEL returns 1.

HP82163 Functions

CLEAR:
Erase the screen.

CLEARO:
Erase the screen starting from the cursor.

---

* Functions are listed in the order they appear in the module.

CSRDN:
Move the cursor once downwards.

CSRHX:
Move the cursor horizontally |X| positions (to the left if X<0, to the right in the opposite case).

CSRL:
Move the cursor once to the left.

CSROFF:
Suppresses the cursor.

CSRON:
Restores the cursor.

CSRR:
Move the cursor once to the right.

CSRVX:
Move the cursor vertically |X| positions (upwards if X<0, downwards in the opposite case).

CSRUP:
Move the cursor once upwards.

CTYPE:
Select cursor type.

HOME:
Reposition cursor to (0,0).

SCRLDN:
Scroll the display one line down.

SCRLUP:
Scroll the display one line up.

SCRLX:
Scroll the display as specified by |X|. Downwards for X<0, upwards for X>0.

XYTAB:
Move the cursor to the position defined by |X| and |Y|.

HP82162 Functions

CLBUF:
Clear the print buffer.

8BIT:
Select 8-BIT mode.

ESCAPE:
Select ESCAPE mode.

PARSE:
Select "line feed on space" mode.

STATUS:
Returns to the X and Y registers 2 bytes indicating the status of the printer.

TABCOL:
Set absolute tabulation at the dot level as defined by |X|.

UNPARSE:
Select "line feed on decimal code 24" mode.

HP82905 Functions

BELL:
Beep.

CHARSET:
Select the character set to use; X=0 for primary, X=1 for secondary.

FFEED:
Form feed one page.

FORMLEN:
Sets number of lines per page as defined by |X|.

GRAPHX:
Indicates to printer that the next |X| bytes of data are graphic.

MODE:
Select printing mode; 0 = Normal, 1 = Expanded, 2 = Compressed, 3 = Bold-expanded, 9 = Bold.

SKIPOFF:
Cancels perforation skips.

SKIPON:
Sets perforation skips.

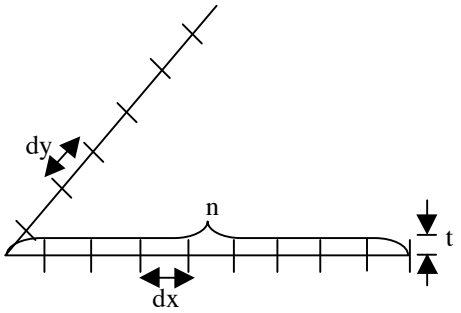
TEXTLEN:
Sets number of text lines per page as defined by |X|.

VSPAC:
Select the line pitch between 2 lines; |X| indicates the number of lines per inch.

Mini-plotter Functions

AXIS:
Draw an axis with the following format:

t = dimension of a half dash -/-
dy = distance between 2 dashes -/- on the y axis
dx = distance between 2 dashes -/- on the x axis
n = number of dashes



Data being provided as follows:
T = t, Z = dy, Y = dx, X = n.

BACKSP:
Move pen one character backward.

BACKSPX:
Move pen backward by |X| characters.

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

COLOR:
Select pen color.

***CSIZE:**
Select character size.

***DRAW:**
Draw a line to (X,Y) coordinates.

***HOME:**
Return pen to coordinates (0,0).

***LABEL:**
Print the contents of the ALPHA register (impression of texts in graphic mode; follows the directions defined by * LDIR).

***LDIR:**
Defines the writing direction for *LABEL.

***MOVE:**
Move the pen to (X,Y) coordinates.

***LTYPE:**
Defines the type of line for * DRAW and RDRAW (|X| = 0 to 15).

***PLREGX:**
A pointer *bbb, eee* being placed in X, traces the broken line passing by the points [(R*bbb*),(R*bbb*+1)], [(R*bbb*+2),(R*bbb*+3)]...[(R*eee*-1),(R*eee*)].

RDRAW:
Draw a line to (X,Y) relative to the current position of the pen.

RESET:
Moves the pen to the left margin and selects Text mode.

REVLF:
Reverse line feed.

REVLFX:
Reverse line feed |X| times.

RMOVE:
Moves the pen to (X,Y) relative to the current position of the pen.

SETORG:
Sets current pen position as the origin (0,0).

Utilities

/MOD:
Return the quotient in Y and the remainder in X of Euclidean division (that is to say calculate the quotient and the modulo of X\Y with integers)

AD-LC:
Returns the coordinates (line,column) of an array element from its address (R*nn*) and the array pointer.

ALENG:
Returns the length of the string in the ALPHA register.

ANUM:
Searches the ALPHA register, for a number. The first number found is returned to the X-register.

ANUMDEL:
In addition to ANUM, It also deletes all characters in the string from the start of the string to the last numerical character used.

APPX:
Appends the integer part of the X-register to the left of the ALPHA register string.

AROT:
Rotates the ALPHA register string by the number of characters specified by the X-register.

ATOXL:
Deletes the first character of ALPHA and returns its decimal code to the X-register.

ATOXR:
Deletes the last character of ALPHA and returns its decimal code to the X-register.

ATOXX:
Returns to the X-register the character of ALPHA specified by the value of the X-register.

BLDPT:
Builds a pointer in X starting from the elements present in Z, Y and X. If X>0, X = *zzz.yyyxx*. If X<0 X = code of matrix such as Z = first register of the table, Y = a number of lines, |X| = a number of columns.

BRKPT:
Break up the X register into three numerical values.

CHFLAG:
During the construction of the program, the user defines a state of the HP-41 by using the usual instructions (in calculation mode). In program mode XEQ " CHFLAG " places in the program 2 lines:

01 CHFLAG
02 "..." unusual alpha string.

During the execution of the program the 2 lines will configure the computer in accordance with the situation defines in the moment of the programming.

CLINC:
Deletes increments in X register (i.e. starting from the 4th digit after the decimal point).

COLPT:
Returns a pointer from the column number in Y and the array pointer in X.

GETRGX:
Copy the data of the current X-memory file where the pointer is to the registers in main memory as defined by the value in the X register.

LC-AD:
Return the address (register number) of an array element from its line and column number.

LINPT:
Return the line pointer of an array knowing the line number and the array pointer.

NOP:
Do not execute any operation.

OUT:
Prefixes intended to facilitate the writing of functions.

POSA:
Returns the numerical position of a character in the ALPHA register specified in the X register.

PSIZE:
Allocates the number of data registers specified by the X-register.

READEM:
Copies the XMEMORY file named in the ALPHA register from a mass storage device. See WRTEM.

RG:
Prefixes to allow easier entry of related functions.

RG+-:
Execute the addition (or the subtraction) of two vectors indicated by the codes placed Y and in X.

RG*:
As RG+-, executes multiplication.

RG/:
As RG+-, executes division.

RG+Y, RG*Y, and RG/Y:
Carry out an arithmetic operation of the elements indicated by X, with the operand placed in Y.

RGAX:
If X<0, it copies the ALPHA register to the data registers in blocks of 6 characters. If X>0, it appends the ALPHA register with the contents of the data registers specified by the X-register.

RGCOPY:
If X>0, copies the data registers indicated by X to the registers specified in Y. If X<0, the registers are swapped. Admits an increment.

RGINIT:
If X>0, places 0 in the registers indicated by X. If X<0, places the numbers 1 to N in the registers.

RGNb:
Return numbers of registers specified by a code (pointer) *bbb.eeeii* in X.

RGORD:
Replace each value, contained in the registers of data specified, by their rank.

RGXTR:
Extract the first register address of the last line in an array.

RGSUM:
Return the sum of the values specified by the code in X If X<0, calculates the sum of the absolute values.

RGVIEW:
Input or display of the registers. Details necessary.

SAVERGX:
Opposite of GETRGX. Recopy the registers indicated in X, in the current data file, starting from the pointer and while following the increment *jj*; X=*bbb.eeeiijj*

SIZE?:
Return the number of registers allocated to the data.

SORT:
Sort by order ascending (X>0) or decreasing (X<0) the values of the registers indicated in X. Sorts alpha and numerical.

STO>L:
Copy the value placed in X with the address specified in L, and increments L.

SUB$:
Extraction and/or justification of strings.

TF55:
Validate an invalid printer.

VKEYS:
Catalog assignments.

WRTEM:
Creates and fills an extended memory file on the cassette. This file is a " WRTA " of extended memory.

X...NN
Functions X=NN?. X≠NN?, X<NN?. X<=NN?, X>=NN? and X>NN? work as the normal test functions but they work between the X-register and any register specified in the Y-register.

X<>F:
Swap the X-register and the "F" register which contains a number that represents flags 0-7.

XTOAL:
Adds the first character of ALPHA and based on its decimal code in the X-register.

XTOAR:
Adds the last character of ALPHA and based on its decimal code in the X-register.

Y/N:
Simplify programs that, during their execution, ask a question for which the answer is YES or NO.

YTOAX:
Replaces the character at X in ALPHA by Y.

82160 (HPIL INTERFACE) FUNCTION GROUP

This group of functions will make the HP82160 HPIL interface easier to use.

These functions work with the HP82160 to get information about the devices on the loop and to send alpha characters and output commands to the devices.

HP-IL protocol provides two ways to identify a device on the loop:

- device ID
- accessory ID

The device ID is a string of data messages that may include the device model number and any other information that the device manufacturer wishes to provide. A Hewlett-Packard device ID is an ASCII-coded character string with the letters "HP" followed by a 5-character model number and a revision letter, and terminated by a carriage return/line feed. For example, the device ID for the HP 82905A printer is the string "HP82905A".

The accessory ID is a numeric code that identifies a device by its general type. The code is one eight-bit byte (decimal value from 0 to 255). Each group of 16 accessory ID numbers forms a device class, whose number (the first four bits of the accessory ID can be obtained by dividing the accessory ID by 16 and taking the integer part. Each entry in a class is specified by the accessory ID modulo 16 (the last four bits of the ID).

Accessory ID		Device Class	
Decimal	Hexadecimal		
0 - 15	0 - F	0	Controller
16 - 31	10 - 1F	1	Mass Storage
32 - 47	20 - 2F	2	Printer
48 - 63	30 - 3F	3	Display
67 - 79	40 - 4F	4	Interface
80 - 95	50 - 5F	5	Instrument
96 - 111	60 - 6F	6	Graphics
112 - 239	70 - EF	7 - 14	Not defined
240 - 255	F0 - FF	15	Future extensions

For example, the accessory ID of the HP82161A Digital Cassette Drive is 16, which makes it entry 0 in the mass storage device class. That is:

```
       Accessory ID = 00010000
                    = 16₁₀
and
       16 MCD 16 = 0
```

These functions are similar to some of the EXTENDED I/O module functions.

AID - Accessory ID -

[AID] returns the Accessory ID of the primary device. The Accessory ID is an integer in the range 0 to 255 that 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, which represents for the Accessory ID of the primary device. To know the AID number of a device, refer to the description of the HPIL message "Send Accessory ID" in the device owner's manual.

If the primary device has no Accessory ID, the X-register is not changed and the error message NO RESPONSE is displayed.

Related functions:

I/O ROM: FINDAID, ID
HPIL ROM: FINDID, SELECT, AUTOIO, MANIO
PANAME ROM: RCLSEL

Appendix C

For every type of accessory, this list gives the name, ID range and number related to a device type. To search for a given accessory type, the accessory type number is put in the X register. This list can be used with FINDAID.

Device class	Accessory ID	Type identifier
Controller	0 to 15	- 1
Mass storage	16 to 31	- 16
Printer	32 to 47	- 32
Display	48 to 63	- 48
Interface	64 to 79	- 64
	80 to 95	- 80
Graphic device	96 to 111	- 96
	112 to 127	-112
	128 to 143	-128
	144 to 159	-144
	160 to 175	-160
	176 to 191	-176
	192 to 207	-192
	208 to 239	-224
	240 to 255	-240

ID - Device ID -

[ID] (Device ID) returns to the ALPHA register a string containing the device ID of the primary device on the HP-IL. The device ID is an alphanumeric string that identifies the device. Generally the ID string indicates the device manufacturer, model number and revision letter.

For instance, the device ID of the HPIL-RS232 interface is "HP82164A". If the main 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 main device to the ALPHA register. To get the ID string of a device, refer to the description of the HPIL message "Send Device ID" in the device owner's manual.

If the main device has no device ID, the ALPHA register is cleared and the HP-41 display shows NO RESPONSE.

ID reads a maximum of 24 characters from the primary device. If the device ID has fewer than 24 characters, the HP-41 terminates input from the device when it received and End-of-Transmission message. Carriage returns (character code 13) and line feeds (code 10) are deleted from the ID string placed in the ALPHA register.

Example: This routine tests whether the primary device is an HP 3468 voltmeter. If it is, the routine returns to a main program. Otherwise, execution halts and the HP-41 displays NOT A VOLTMETER.

```
01 LBL "VOLT?"            Is primary device a voltmeter?
02 SF 25
03 ID                     Get device ID.
04 FC?S 25
05 GTO 01                 Branches if devices does not return a device ID.
06 ASTO X                 Put ID into X.
07 "HP3486"               Look at first 6 characters in ID.
08 ASTO Y
09 X=Y?                   Is primary device an HP3486?
10 RTN                    Yes.
11 LBL 01
12 "NOT A VOLTMETER"      Displays an error.
13 AVIEW
14 STOP
```

Related functions:

HPIL ROM: FINDID, SELECT, AUTOIO, MANIO
PANAME ROM: AID, FINDAID, RCLSEL

FINDAID - Find an Accessory ID -

[FINDAID] (FIND Accessory ID) Allows the HP-41 to locate a device of a specific class or type which you specify with a number in the X-register.

INSTRUCTIONS FOR FINDAID

To locate a specific device, place the device's ID or type number in the X-register. Execute FINDAID; an integer number, which represents the address of the located device is returned to the X-register.

If the number you entered is positive, FINDAID searches the loop starting with the primary device until it finds a device whose accessory ID matches the number (integer part) in the X-register. If such a device is present, its HP-IL address is returned in the X-register.

If the number you enter in the X-register is negative, FINDAID searches the loop in the same manor as described above. However, the value returned is the address of the first device found in the device class corresponding to the absolute value of the number in the X-register. Recall from the discussion that

device class = INT[ABS(n/16)].

Thus, when n is negative, FINDAID matches only the first four bits of the accessory ID with the first (leftmost) four bits of the number in the X-register.

If there are no devices of the specified type or class present, 0 is placed in the X-register. DATA ERROR is displayed if FINDAID is executed with an input outside the range $-255 \leq n \leq 255$.

FINDAID can be used to find the address of a desired device and then SELECT that device as that primary device in order to send information to it.

EXAMPLE: FIND? Returns the address of the first graphics device, starting from the primary device.

```
01 LBL FIND? Find a plotter
02 -96       Look for a graphics device
03 FINDAID
04 RTN
```

OUT - Input prefix for OUT functions -

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.

ASN "OUT" 15

Keystrokes: [ASN][ALPHA][O][U][T][ALPHA][1][5] Put the calculator in USER mode, then execute or program a function which 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 letters.
...(for instance YBX for the OUTYBX function).

[ALPHA]

OUTAX - OUTA with repetition according to X -

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

If flag 17 is cleared, an End Line sequence is added to the ALPHA string every time this one is sent on the HPIL loop. (End Line: characters CR and LF, decimal code 13 and 10).

If Flag 17 is set, the ALPHA string is sent several times 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 "_*" (stands for the SPACE character) strings on a HP82905B printer, use the following sequence. (The printer must be the main device)

"_*" SF 17 40 OUTAX ADV

Related functions:

HPIL ROM: OUTA, MANIO, SELECT are used to select the device.

OUTCR - OUTput Carriage Return -

[OUTCR] send a CR character to the main device. (Carriage Return: decimal code 13).

OUTLF - OUTput Line Feed -

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

OUTLFX - OUTput Line Feeds by X -

[OUTLFX] sends one or several LF characters to the main 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 space characters -

[OUTSPX] (OUTput SPaces by X) sends the number of space characters (decimal code 32) specified by the absolute value of the X-register, (0<=X<=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>Y?   GTO 01
-   LBL 02   SF 17   OUTA   OUTSPX   RTN
LBL 01   DSE Y   NOP   CLX   1 E2   /   SUB$
CLX   GTO 02   END
```

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

(1): In this case, the ALPHA register is modified by OUTAT.

OUTXB - Send a character by its decimal code -

[OUTXB] sends to the primary device, the character, which 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 send to the printer the character "\" (Decimal code 92), use the sequence: 92 OUTXB.

OUTYBX - Send a character, several times, by its decimal code -

---

[OUTYBX] sends to the primary device, one or several times, a character which 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 send.

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 Zero)
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 - OUTA with 7th bit set -

[OUTa] works like OUTA except that the 7th bit of every character sent is set. Therefor 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 "inverse 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 inverse 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 to use with the OUTa function.

OUTaX - 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 send 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 "inverse video" on the HP82163 video interface (which has to be the primary device), use the following sequence:

"-*" SF 17 16 OUTaX

RCLSEL - Recall primary device address -

[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 with a 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, which is 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 returned 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 (VIDEO INTERFACE) FUNCTION GROUP

This group of functions will make the HP82163 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 owner's 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 one can use these functions with video interfaces, such as the Mountain Computer MC00701A (AID 50), PAC-TEXT (AID 48), or KRISTAL* MINITEL interface (AID 48).

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

* KRISTAL, Chemin des Clos Zirst 38240 MEYLAN (FRANCE), information processing systems interfaces and technical applications, instrumentation and OEM approved HP ICC.

CLEAR - 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 identity of the HP 82905B is 33 (or 48 per the French manual).

CLEARO - CLEAR the display from the cursor On -

[CLEARO] (CLEAR from cursor On) 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.

CSRDN - Move cursor down -

[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.

CSRHX - Move cursor Horizontally by X -

[CSRHX] (move CurSoR Horizontally by X) moves the cursor horizontally. The absolute value of X specifies the number of characters of the move and its sign determines 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 moves, then execute CSRHX.

* This is not true for every non-HP video interface.

CSRL - 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.

CSROFF - 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-).

CSRON - Display the cursor -

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

CSRR - 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.

CSRVX - Move the cursor down according to X -

[CSRVX] (move CurSoR Vertically by X) move the cursor vertically. The absolute value of X specifies the number of lines of the move and its sign determines 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.

CSRUP - Move the cursor up -

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

CTYPE - Select the type of cursor -

[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.

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

---

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

SCRLDN - Scroll the display down -

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

SCRLUP - Scroll the display one line up -

[SCRLUP] (SCRolL 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.)

SCRLX - Scroll the display according to X -

[SCRLX] (SCRolL as specified by X) the display is scroll according to X. The absolute value of X specifies the number of lines of the scroll, and its sign the direction.

- For X<0 SCRLX performs (-X) SCRLUPs. (Scrolls the display up by (-X) lines.)
- For X>=0, SCRLX performs X SCRLDNs. (Scrolls the display down by X lines.)

INSTRUCTIONS FOR SCRLX

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

XYTAB - Move the cursor to position (X,Y) -

[XYTAB] ((X,Y) TABulate) moves the cursor to position (x,y), The column number is specified by the absolute value of X and the line number by that of Y.

INSTRUCTIONS FOR XYTAB

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

Appendix V

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

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

Function(s)	Sequence	Characters codes
CLEAR	ESC E	27 69
CLEARO	ESC J	27 74
CSRDN, CSRVX for X>=0	ESC B	27 66
CSRL, CSRHX for X<0	BS	08
CSROFF	ESC <	27 60
CSRON	ESC >	27 62
CSRR, CSRHX for X>=0	ESC C	27 67
CSRUP, CSRVX for X<0	ESC A	27 65
CTYPE for X=0	ESC Q	27 81
CTYPE for X=1 or -1	ESC R	27 82
HOME	ESC H	27 72
SCRLDN, SCRLX for X>=0	ESC T	27 84
SCRLUP, SCRLX for X<0	ESC S	27 83
XYTAB	ESC % {c} {l}	27 37 col ln

82162 (THERMAL PRINTER) FUNCTION GROUP

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

These features are:

- Two different sets of characters;
- A "parse" mode;
- An absolute dot-level tabulation function, independent from any data yet 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.

8BIT

- Select "8 bit mode" -

[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.

ESCAPE

- Select "Escape" mode -

[ESCAPE] selects the "escape" mode, which activates the ASCII (not-HP41) character set. In this mode, you cannot use specific printing functions to send characters to the printer because they automatically select the "8 bits mode". However some applications may require the use of the ASCII character set. The ESCAPE function enables the utilization 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.

PARSE

- Select "Line feed on space" mode -

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

CLBUF

- Clear the buffer -

[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 - Select the "line-feed on the 24th character mode" -

[UNPARSE] disables the special mode selected by UNPARSE.

TABCOL - Column tabulation -

[TABCOL] enables an absolute tabulation on the dot level as opposed to SKYCOL, 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 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
```

STATUS - Recall printer status -

[STATUS] returns to the Y-register an integer, which specifies the primary device's first status bit, and in the X-register an integer, which specifies the primary device's second status bit. The effect of STATUS on the stack depends whether stack lift is enabled or not when the function is executed:

- If stack lift is enabled:

Before	After
T: t	T: y
Z: z	Z: x
Y: y	Y: 1st status bit
X: x	X: 2nd status bit

- If stack lift is not enabled:

Before	After
T: t	T: z
Z: z	Z: y
Y: y	Y: 1st status bit
X: x	X: 2nd status bit

However, the LASTx-register is not modified.

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

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

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

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

To know the number, and definition of the status bits of a device, refer to the description of the HP-IL message "SEND STATUS" in the device manual.

The S1 appendix gives the detailed definition of the two status bits of the HP82162A printer.

Appendix S1

Printer Status Byte Definitions.

	BIT NUMBER	BIT VALUE*	NAME	DEFINITION
FIRST BYTE	7	-	-	Always set to 0.
	6	64	SR	Service request. Set for out of paper condition, carriage jam, or PRINT or PAPER ADVANCE key pressed. Cleared when condition removed or by Send Status, Device Clear, Selected Device Clear, or Loop Powder Down message.
	5	32	MB }	Print mode/jam. Set according to:
	4	16	MA }	MB MA Condition 0 0 Manual print mode 0 1 Trace print mode 1 0 Normal print mode 1 1 Carriage jam (valid only if ER = 1)
	3	8	ER	Error condition. Set out-of-paper or carriage jam.
	2	4	PA	Paper Advance. Set while PAPER ADVANCE key is down.
	1	2	PR	Print. Set while PRINT key is down.
	0	1	LA	Long advance. Set following out-of-paper condition, indicating long carriage movement during advance. Cleared when Carriage Return character received.
SECOND BYTE	7	128	EL	End of line. Set if carriage return was last byte received.
	6	64	ID	Idle. Set if printer is not printing.
	5	32	BE	Buffer empty. Set if no information is accumulated in print buffer.
	4	16	EB	Eight bit. Set if printer is in Eight-Bit mode.
	3	8	RJ	Right justify. Set if printer is set to right-justify mode.
	2	4	DW	Double wide. Set if printer is set to double-wide mode.
	1	2	CO	Column mode. Set if printer is set to column mode.
	0	1	LC	Lowercase. Set if printer is in Eight-Bit mode and set to lowercase mode

* Add the bit values for all bits that are set to "1"} to find the decimal value of the status byte.

82905 (80-COLUMN PRINTER) FUNCTION 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 permit 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 information on sequences sent by these functions, refer to Appendix P.

BELL - Beep signal -

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

CHARSET - Character set selection -

[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 information on both character sets.

FFEED - Form feed -

[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.

FORMLEN - Page length -

[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 re-initialization with the CLEAR function (refer to this function for further information) the default line count is 66.

GRAPHX - Graphic output -

[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 Manuel to find the relations between data sent and printer output (Graphic mode paragraph).

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

MODE - Printing mode -

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

X value:	Mode:	No. char./line:
0	Normal	80
1	Expanded	40
2	Compressed	132
3	Bold-expanded	66
9	Bold	80

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

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

SKIPOFF - Cancel perforation skip -

SKIPOFF cancels the SKIPON function.

SKIPON - Set perforation skip -

[SKIPON] sets 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 lines per page is selected with the TEXTLEN function). So nothing is printed on the perforations.

Perforations skip mode is off, at power on or after using the CLEAR function. (Refer to this function for other information).

TEXTLEN - Text length -

[TEXTLEN] sets the number of text lines per page according to the absolute value of X. This number must be in the range 1 to the number of lines per page (selected with FORMLEN). At power on or after using the CLEAR function (refer to this function for the description of this feature), the default lines number per page is 60.

VSPAC - Vertical space -

[VSPAC] selects the vertical space in line 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 return DATA ERROR.

Appendix P

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

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

Function(s)	Sequence	Codes	Thinkjet
BELL	BEL	07	
CHARSET for X=0	SI	15	Normal
CHARSET for X=1 or -1	SO	14	Bold
FFEED	FF	12	FF
FORMLEN	ESC &l {#} P	27 38 108 {par} 80	FL
GRAPHX	ESC *b {#} G	27 42 98 {par} 71	
MODE	ESC &k {#} S	27 38 107 {par} 83	0 Normal (80 c/l) 1 expanded (40 c/l) 2 compressed (142 c/l) 3 expanded-compressed (71 c/l) 9 bold (80 c/l)
SKIPOFF	ESC &l0L	27 38 108 48 76	skipoff
SKIPON	ESC &l1L	27 38 108 49 76	skipon
TEXTLEN	ESC &l {#} F	27 38 108 {par} 70	textlen
VSPAC	ESC &l {#} D	27 38 108 {par} 68	vspac

Roman-8 Characters

CHAR.	DEC.	HEX.
CTL@	0	00
CTLA	1	01
CTLB	2	02
CTLC	3	03
CTLD	4	04
CTLE	5	05
CTLF	6	06
CTLG	7	07
CTLH	8	08
CTLI	9	09
CTLJ	10	0A
CTLK	11	0B
CTLL	12	0C
CTLM	13	0D
CTLN	14	0E
CTLO	15	0F
CTLP	16	10
CTLQ	17	11
CTLR	18	12
CTLS	19	13
CTLT	20	14
CTLU	21	15
CTLV	22	16
CTLW	23	17
CTLX	24	18
CTLY	25	19
CTLZ	26	1A
CTL[	27	1B
CTL\	28	1C
CTL]	29	1D
CTL^	30	1E
CTL_	31	1F

CHAR.	DEC.	HEX.
	32	20
!	33	21
"	34	22
#	34	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F

CHAR.	DEC.	HEX.
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
[	91	5B
\	92	5C
]	93	5D
^	94	5E
_	95	5F

CHAR.	DEC.	HEX.
`	96	60
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
{	123	7B
|	124	7C
}	125	7D
~	126	7E
¦	127	7F

MINIPLOTTER FUNCTION GROUP

Several Miniplotters can be used with the HP-41. TANDY, CANON... have the same mechanics and the same command set. Of course, the miniplotter should be interfaced with the HP-IL loop with the HP82166A GP IO interface.

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

These miniplotters main characteristics are:

- 4 colors. The print head is in fact a set of 4 mini ball pen. The color can be changed, either by program or by a switch, during plotting.

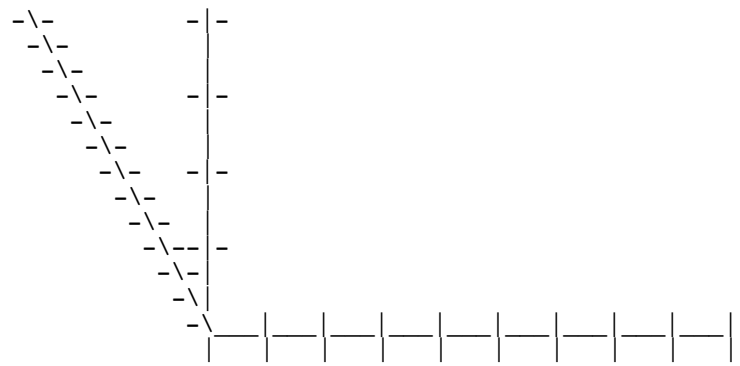
- 11.4 cm paper, in roll. It is possible to draw, or plot on the length of the paper, so one can print large charts. Refer to the EXAMPLE.

- 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 outdated too quickly.

AXIS - 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 dash
Z: vertical distance between two dashes
Y: horizontal distance between two dashes
X: number of dashes

The axis is drawn from the current pen position, and the direction depends of the values in the Y and Z registers. However, dashes are always either vertical or horizontal according to the axis inclination from the horizontal (X direction): under 45 degrees, dashes 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 a 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              Re-initialization
06 "No. COL. ?"
07 PROMPT             Input the number of columns (C).
08 STO 00             R00= Number of columns.
09 "COL. WIDTH ?"
10 PROMPT             Input the column's width.
11 STO 01             R01= Column's width.
12 *                  First chart dimension.
13 "HT. LINE ?"
14 PROMPT             Input the line height.
15 STO 02             R02= Height of each line.
16 ST+ X              There are 2 lines, the 2nd dimension is 2*X.
17 CHS                The displacement will be down!
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
```

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

Starting position.

Drawing of inside lines.

```
XEQ "CHART"
No. COL. ?
4.0000 RUN
COL. WIDTH ?
100.0000 RUN
HT. LINE ?
50.0000 RUN
```

Will print:

*LDIR - Writing direction in graphic mode -

[*LDIR] (*Line DIRection) specifies the writing direction for *LABEL. There are four possibilities:

0 to the right, 1 down, 2 to the left, 3 up.

*LTYPE - Line Type -

*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 - Move pen up -

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

*PLREGX - Broken line plotting -

[*PLREGX] (PLot REGisters by X) joins the points, which 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 go to the next coordinate (numeric data) without plotting (*MOVE), there it resumes plotting (DRAW).

RDRAW - Relative Drawing -

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

RESET - Re-initialization -

[RESET] moves the pen to the left margin and selects Text mode.

REVLF - Reverse line feed -

[REVLF] (REVerse Line Feed) moves the pen one line upward.

REVLFX - Reverse line feed by X -

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

RMOVE - Relative MOVEment -

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

SETORG - Set origin -

[SETORG] (SET ORiGin) sets the current pen position has the origin (0,0).

BACKSP - BACKSPace -

[BACKSP] moves the pen one character backward.

BACKSPX - 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 - Box drawing -

[BOX] draws a rectangle, which 2 opposite angles have the coordinates:

(x1,y1) and (x2,y2), with T=y2, Z=x2, Y=y1, X=x1.

COLOR - Color selection -

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

*CSIZE - Character size -

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

*DRAW - Draw a segment -

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

*HOME - Send pen to origin -

[*HOME] sends pen to (0,0) coordinates.

*LABEL - Print the ALPHA register -

[*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 with the *LDIR function.

Appendix 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, eventually with a sign and no more than 4 digits
(For instance: -230; 0024);

The syntax column specifies the signification of each parameter;

Control characters (Decimal values):

17: Select TEXT mode
18: Select "GRAPH" mode
11: Reverse line feed (Text mode only)
08: Backspace (Text mode only)

GRAPHIC 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).

GRAPHIC mode functions

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

Unspecific mode functions

These mini-plotter instructions must be executed in Graphic mode. So these functions set Graphic mode before executing the instruction. However these functions are often used in Text mode. The user can control in which mode the mini-plotter is left after execution.

UTILITIES

These functions and routines have a wide range of applications:

- Manipulation of numeric and alphanumeric arrays (one or two dimensions):

- Numeric or alphanumeric sorting;

- Character string manipulation:

- Extended memory management (HP82180A XFUNCTIONS and HP82181A XMEMORY);

- Wide range of other applications...!

The pointer used with the array handling functions is like the control number described in the Main Memory section in the HP-41 manual. It's in the format *bbb.eeeii* where *bbb* is the *beginning* of the main memory register block, *eee* is the *end* of the main memory register block, and *ii* is the increment or number of registers to skip inclusive of the current register. For example, pointer 30.03402 specifies that the first register is 30, the last register is 34 and the increment is 2, therefore, registers 30, 32, and 34 will be selected. The *ii* value could also represent the number of columns of an array.

Some of the functions in this section are similar or identical to their X-FUNCTION counter part. Detailed coverage of those functions is beyond the scope of this manual. It is assumed that you have familiarity with these functions or have the reference materials available to you to review these functions.

- ARRAY HANDLING FUNCTIONS -

BLDPT

- Build a pointer -

[BLDPT] (BuiLD PoinTer) builds a data set 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 calculate the data set 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 nº 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

	column				
	nº1	nº2	nº3	nº4	nº5
line nº1	R25	R26	R27	R28	R29
line nº2	R30	R31	R32	R33	R34
line nº3	R35	R36	R37	R38	R39
line nº4	R40	R41	R42	R43	R44

ARRAY A

INSTRUCTIONS FOR BLDPT

1) To build a *bbb.eeeii* data set pointer where *bbb* specifies the address of the first register of the data set, *eee* specifies the last register of the data set and *ii* is the number of registers between sets:

- 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].

Note: 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:

for X>0		For X<0	
Input:	Output:	Input:	Output:
T: t	T: t	T: t	T: t
Z: *bbb*	Z: t	Z: *bbb*	Z: t
Y: *eee*	Y: t	Y: *lll*	Y: t
X: *ii*	X: *bbb.eeeii*	X: *cc*	X: *bbb.eeecc*
L: I	L: *eee*	L: I	L: *lll*

BRKPT - Break pointer -

[BRKPT] (BReaK PoinTer) breaks 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 registers 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).

	column				
	nº1	nº2	nº3	nº4	nº5
line nº1	R25	R26	R27	R28	R29
line nº2	R30	R31	R32	R33	R34
line nº3	R35	R36	R37	R38	R39
line nº4	R40	R41	R42	R43	R44

ARRAY A

INSTRUCTIONS FOR BRKPT

1) To break 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 check 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*)/*ccc* to the Y-register, and the number of columns (*cc*) to the X-register.
The array pointer is saved in LASTx.

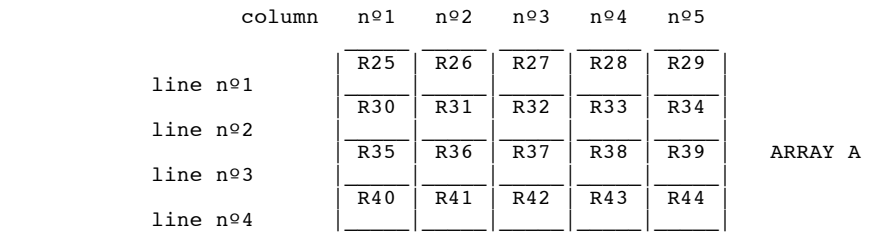
Note: If there is an Alpha string in the X-register, ALPHA DATA is displayed.

STACK :

for X>0		For X<0	
Input:	Output:	Input:	Output:
T: t	T: x	T: t	T: x
Z: z	Z: *bbb*	Z: z	Z: *bbb*
Y: y	Y: *eee*	Y: y	Y: *lll*
X: *bbb.eeeii*	X: *ii*	X: *bbb.eeecc*	X: *cc*
L: l	L: *bbb.eeeii*	L: l	L: *bbb.eeecc* where *eee*=(ll*|*cc*|)-1+*bbb*

[AD-LC] (ADdress-Line-Column) returns the coordinates (line,column) of an array element from its address (R*nn*) and the array pointer.

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




```
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.
[LASTx]                 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.

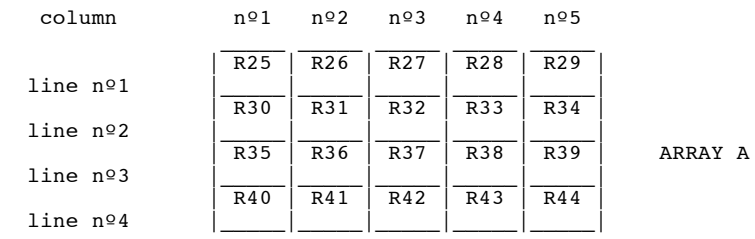
STACK

```
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
```

Note: 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.

[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, with 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 calculate 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 places the pointer in LASTx.

STACK:

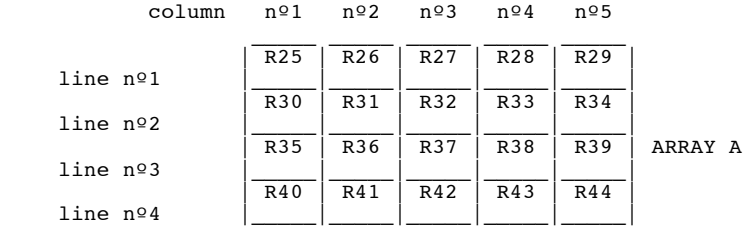
Input:	Output:
T: T	T: T
Z: line nº	Z: T
Y: column nº	Y: T
X: Array Pointer	X: register nº
L: L	L: Array pointer

COLPT - 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 gets the second column numbers of the array A, which pointer is in register 00.

```
Keystrokes:              Display:

2                        2_            Column number.
[RCL] 00                 25.04405      Recall pointer.
[XEQ] "COLPT"            26.04105      2nd column pointer.
```



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:

```
Input :                   Output :

T: t                      T: t
Z: z                      Z: t
Y: Column Nº              Y: z
X: bbb.eeeii              X: b'b'b'.e'e'e'i'i'

L: l                      L: bbb.eeeii
```

Note: *i'i'=ii*

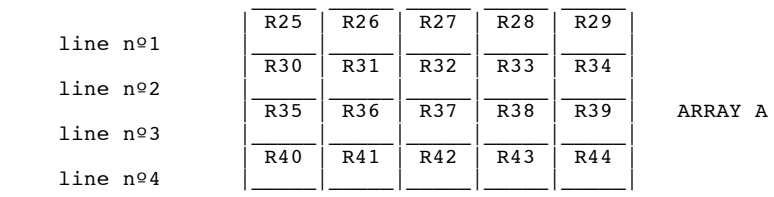
LINPT

- 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 know the registers used by the 2nd line of array A, which array pointer is saved in register R00:

```
Keystrokes:              Display:

[2]                       2_              Line number.
[RCL] 00                  25.04405        Recall array pointer.
[XEQ] "LINPT"             30.03400        2nd line pointer.
```



INSTRUCTIONS FOR LINPT

1) Input the line number, which 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:

```
Input:                    Output:

T: t                      T: t
Z: z                      Z: t
Y: line Nº                Y: z
X: bbb.eeeii              X: b'b'b'.e'e'e'i'i'

L: l                      L: bbb.eeeii
```

CLINC - Clear Increment -

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

Example: You want to know the first and last registers of an array.
The array pointer 25.04405 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	
[EEX] 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 register L.

STACK

Input:	Output:
T: t	T: t
Z: z	Z: z
Y: y	Y: y
X: *bbb.eeeii*	X: *bbb.eee*
L: l	L: *bbb.eeeii*

Note: If the X-register contains an Alpha string, ALPHA DATA is displayed.

GETRGX

- Recall registers from X-memory -

[GETRGX] (GET ReGisters by X) copies in the registers specified by the X-register, the data of the current X-MEMORY file, starting from pointer position, and according to the increment in the X-register.

Example: The pointer in the current X-MEMORY file "DATA" 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 current X-Memory file to the main memory registers as specified by the pointer in the X-register.

STACK:

The stack is unchanged by [GETRGX].

APPLICATION PROGRAMS

The figure 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.

[*] Part of the Extended Functions found in the X FUNCTIONS module or built into the HP-41CX.

FIGURE 1. DATA

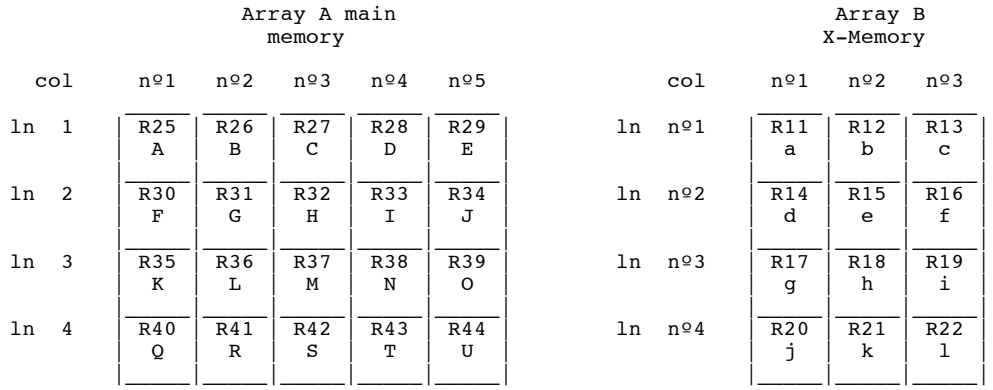
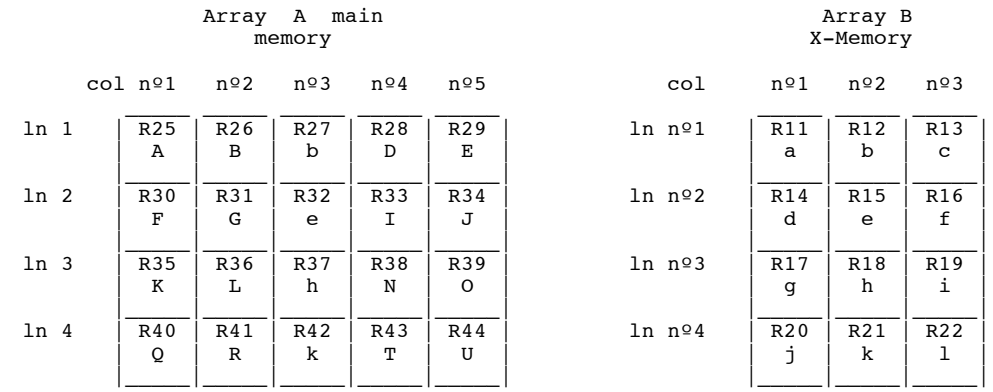


FIGURE 2. RESULTS (AFTER [GETRGX])



SAVERGX - Save registers to X-memory -

[SAVERGX] (SAVE ReGisters by X), copies the specified registers to the current data file in X-MEMORY starting from the pointer and following the increment all controlled by X; X=*bbb.eeeiijj*

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

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. *bbb* is the first main memory register and *eee* is the last main memory register from which you want to copy the data set to the X-Memory file, *ii* is the increment for the main memory registers, and *jj* the increment for the X-Memory registers.

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

STACK:

The stack is unchanged by [GETRGX].

APPLICATION PROGRAMS

The figure below represent 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 3rd column of array A in main memory to the second column of the array B in X-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] "SAVERGX" copies the registers as specified by the pointer in the X-register; the result is represented on the second figure.

[*] Part of the Extended Functions found in the X FUNCTIONS module or built into the HP-41CX.

FIGURE 1. DATA

Array A main memory

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

Array B X-Memory

col	nº1	nº2	nº3
ln nº1	R11 a	R12 b	R13 c
ln nº2	R14 d	R15 e	R16 f
ln nº3	R17 g	R18 h	R19 i
ln nº4	R20 j	R21 k	R22 l

FIGURE 2. RESULTS (AFTER [SAVERGX])

Array A main memory

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

Array B X-Memory

col	nº1	nº2	nº3
ln nº1	R11 a	R12 B	R13 c
ln nº2	R14 d	R15 E	R16 f
ln nº3	R17 g	R18 H	R19 i
ln nº4	R20 j	R21 K	R22 l

[SORT] (SORTer) sorts the registers specified in the X-register.

Example: In the array A below:

1) Sort in increasing order, the 2nd column values.

2) Sort in decreasing order, the 3rd column values.

column	nº1	nº2	nº3	nº4	nº5	
	R25	R26	R27	R28	R29	
line nº1	14	B	21	2	8	
	R30	R31	R32	R33	R34	
line nº2	7	13	19	20	1	
	R35	R36	R37	R38	R39	ARRAY A
line nº3	0	A	-12	18	24	
	R40	R41	R42	R43	R44	
line nº4	23	99	50	11	17	

Keystrokes :	Display :	
2 [RCL] 00 [COLPT]	26.04105	Build 2nd column pointer.
[XEQ] "SORT"	SORTING	Sort is performed.
	26.04105	Done.
3 [LASTx] [COLPT] [CHS]	-27.04205	3rd column pointer; the negative pointer indicates
		a decreasing order.
[XEQ] "SORT"	SORTING	Sort is performed.
	-27.04205	Done.

column	nº1	nº2	nº3	nº4	nº5	
	R25	R26	R27	R28	R29	
line nº1	14	13	50	2	8	
	R30	R31	R32	R33	R34	
line nº2	7	99	21	20	1	
	R35	R36	R37	R38	R39	ARRAY A
line nº3	0	A	19	18	24	
	R40	R41	R42	R43	R44	
line nº4	23	B	-12	11	17	

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 higher than numeric values.

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

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

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

5) While sort is done, if there is no message in the display, "SORTING" is displayed.

STACK:

[SORT] leaves the stack unmodified.

[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 not done.

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

```
Keystrokes :              Display :

1                         1_
[RCL] 00                  25.04405      Recall the pointer.
[LINPT]                   25.02900      Calculate 1st line pointer.
[STO] [.] [L]             25.02900      Store it in the L-register.
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"             60.0000       Store 2nd element.
70                        70_
[XEQ] "STO>L"             70.0000
80                        80_
[XEQ] "STO>L"             80.0000
90                        90_
[XEQ] "STO>L"             90.0000
[LASTx]                   30.0290
```

INSTRUCTIONS FOR STO>L

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

[STOL>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, so programs need less memory.

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
```

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

Note: 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 in the middle of a program. So to input, the 1st column of the array B, which 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
```

column	nº1	nº2	nº3	nº4	nº5	
	R25	R26	R27	R28	R29	
line nº 1	50					
	R30	R31	R32	R33	R34	
line nº 2	60					
	R35	R36	R37	R38	R39	ARRAY A
line nº 3	70					
	R40	R41	R42	R43	R44	
line nº 4	80					

[RG] is a function, which makes easier the entry of functions beginning with "RG". This function should be assigned to a key. For instance, assign [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", hit 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 first 2 letters.
...(ex. SUM for RGSUM).

[ALPHA]

RGAX - 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 the registers specified by the register pointer in the X-register, to the ALPHA 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, you want to send, registers R12 and R16, to the ALPHA register:

```
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 appends an end 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 remain in the ALPHA register. The leftmost characters are lost. Loading stops when an end 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:

```
Input:              Output:
T: t                 T: t
Z: z                 Z: z
Y: y                 Y: y
X: Initial pointer   X: New pointer

L: l                 L: Initial pointer
```

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

If X>=0, [RGCOPY] copies the registers specified by the X-register pointer, to the registers specified by the Y-register pointer.

If X<0, [RGCOPY] swaps the registers specified in the X-register, and the one specified in the Y-register.

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

```
          column    nº1   nº2   nº3   nº4   nº5
                   ┌─────┬─────┬─────┬─────┬─────┐
                   │ R25 │ R26 │ R27 │ R28 │ R29 │
        line nº1   │  1  │  2  │  3  │  4  │  5  │
                   ├─────┼─────┼─────┼─────┼─────┤
                   │ R30 │ R31 │ R32 │ R33 │ R34 │
        line nº2   │  6  │  7  │  8  │  9  │  10 │
                   ├─────┼─────┼─────┼─────┼─────┤
                   │ R35 │ R36 │ R37 │ R38 │ R39 │    ARRAY B
        line nº3   │  11 │  12 │  13 │  14 │  15 │
                   ├─────┼─────┼─────┼─────┼─────┤
                   │ R40 │ R41 │ R42 │ R43 │ R44 │
        line nº4   │  16 │  17 │  18 │  19 │  20 │
                   └─────┴─────┴─────┴─────┴─────┘
```

Suppose that the array pointer is in register R00.

```
Keystrokes :              Display :

3 [RCL] 00 [COLPT]         27.04205     Destination pointer.
1 [LAST X] [COLPT]         25.04005     Origin pointer.
[XEQ] "RGCOPY"             27.04205     Destination pointer.
[RGVIEW]                                list the 3rd column R27= 1,..., R42= 16.
1 [RCL] 00 [LINPT]         25.02900     1st Pointer.
2 [LAST X] [COLPT] [CHS] -26.04105     2nd pointer.
[RG] "COPY"                25.02900     The stack moved down.
```

The final array is:

```
          column    nº1   nº2   nº3   nº4   nº5
                   ┌─────┬─────┬─────┬─────┬─────┐
                   │ R25 │ R26 │ R27 │ R28 │ R29 │
        line nº1   │  2  │  7  │  12 │  17 │  5  │
                   ├─────┼─────┼─────┼─────┼─────┤
                   │ R30 │ R31 │ R32 │ R33 │ R34 │
        line nº2   │  6  │  1  │  6  │  9  │  1O │
                   ├─────┼─────┼─────┼─────┼─────┤
                   │ R35 │ R36 │ R37 │ R38 │ R39 │    ARRAY B
        line nº3   │  11 │  1  │  11 │  14 │  15 │
                   ├─────┼─────┼─────┼─────┼─────┤
                   │ R40 │ R41 │ R42 │ R43 │ R44 │
        line nº4   │  16 │  4  │  16 │  19 │  20 │
                   └─────┴─────┴─────┴─────┴─────┘
```

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, so that no information is lost.

STACK:

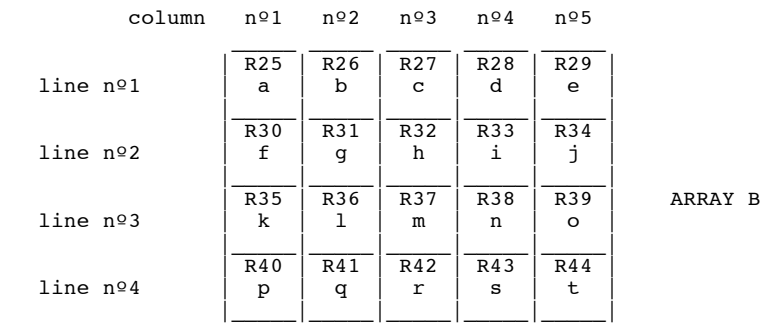
```
Input:                      Output:

T: t                        T: t
Z: z                        Z: t
Y: Destination pointer      Y: z
X: Origin pointer           X: Destination pointer

L: l                        L: Origin pointer
```

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

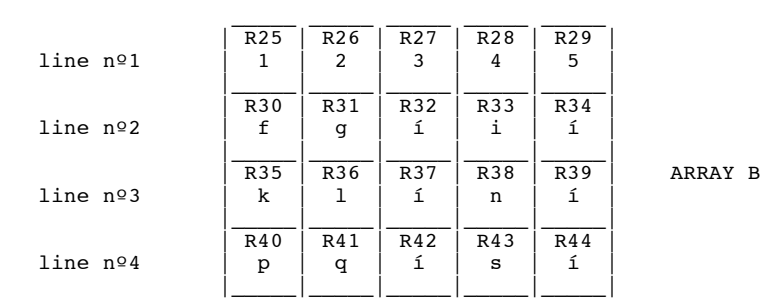
If X>=0 [RGINIT] puts zero in all the registers specified by the pointer in the X-register.

If X<0 [RGINIT] puts integers, from 1 to N, in the registers specified by the pointer in the X-register.

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, throughout the first line.




```
Keystrokes :              Display :
3 [RCL] 00 [COLPT]           27.04205      3rd column pointer.
[XEQ] "RGINIT"               27.04205      Initialize 3rd column to zero.
5 [LASTx] [COLPT]            29.04405      5th column pointer.
[XEQ] "RGINIT"               29.04405      Initialize 5th column to zero.
1 [LASTx] [LINPT] [CHS]     -25.02900      Negative sign to indicate an
[XEQ] "RGINIT"              -25.02900      initialization with integers 1 to N.
```



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].

RGNb

- Number of register by pointer -

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

Example: To know the number of elements of an array, which pointer is saved in register R00; then to know the number of register per line.

```
Keystrokes :                Display :

[RCL] 00 [CLINC]            25.04400    Registers pointer.
[XEQ] "RGNb"                20.00000    The array contains 20 registers.
1 [RCL] 00 [LINPT]          25.02900    Line pointer.
[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 in 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
```

RGORD - Replace values in order of their ranking -

Disclaimer: The following discussion is the result of spectulation and experimentation. The original information on this function was from the one sentence description in the FUNCTION OVERVIEW section of the French manual. We will begin with the theory of how the function should work followed by the actual experimental findings.

[RGORD] (ReGister ORDer) replaces register data with their order.

Replaces the data (D) in selected registers (Rnn), specified by the pointer in the X-register, by their number in the hierarchical order of the registers selected (N) based on the data value contained in each register. 1 is placed in the register with the highest value, 2 in the next highest down to N in the lowest.

Unlike SORT, this function doesn't arrange the registers or data based on the values in the registers, it replaces the data in the registers with integer values in the respective registers.

Example: Starting with array B below, to determine what the hierarchical order of the data in line 2 is, start with LINPT to get the pointer 30.03400 in the X-register then execute RGORD.

column	nº1	nº2	nº3	nº4	nº5	
	R25	R26	R27	R28	R29	
line nº1	-14	15	21	2	8	
	R30	R31	R32	R33	R34	
line nº2	7	13	19	-20	1	
	R35	R36	R37	R38	R39	ARRAY B
line nº3	0	6	12	18	24	
	R40	R41	R42	R43	R44	
line nº4	23	4	5	11	17	

The resulting array is:

column	nº1	nº2	nº3	nº4	nº5	
	R25	R26	R27	R28	R29	
line nº1	-14	15	21	2	8	
	R30	R31	R32	R33	R34	
line nº2	3	2	1	5	4	
	R35	R36	R37	R38	R39	ARRAY B
line nº3	0	6	12	18	24	
	R40	R41	R42	R43	R44	
line nº4	23	4	5	11	17	

As another example, to determine the order of a specific column, say the 4th column, you could start by using COLPT to generate 28.04305 in the X-register. The result after RGORD is:

```
        column     nº1    nº2    nº3    nº4    nº5
                  ┌─────┬─────┬─────┬─────┬─────┐
                  │ R25 │ R26 │ R27 │ R28 │ R29 │
line nº1          │ -14 │  15 │  21 │   3 │   8 │
                  ├─────┼─────┼─────┼─────┼─────┤
                  │ R30 │ R31 │ R32 │ R33 │ R34 │
line nº2          │   7 │  13 │  19 │   4 │   1 │
                  ├─────┼─────┼─────┼─────┼─────┤
                  │ R35 │ R36 │ R37 │ R38 │ R39 │    ARRAY B
line nº3          │   0 │   6 │  12 │   1 │  24 │
                  ├─────┼─────┼─────┼─────┼─────┤
                  │ R40 │ R41 │ R42 │ R43 │ R44 │
line nº4          │  23 │   4 │   5 │   2 │  17 │
                  └─────┴─────┴─────┴─────┴─────┘
```

STACK:

The stack is unchanged by the execution of [RGORD].

Now is time for the experimental findings.

When first experimenting with this function, I used RGINIT to put 1, 2,...N into the registers so I would have some quick data to mess with. Since the data was positive and in ascending order, the function seemed to work fine. It was only when I executed the function a second time in succession on the results it had generated after the initialization that I noticed inconsistencies. Note below the data in descending order and the resulting values after the execution of the data.

Data	16	14	12	10	8
Expected results	1	2	3	4	5
Actual results	4	5	2	3	1

The results get even more outrageous when negative numbers are part of the data:

Data	-20	-30	12	10	8
Expected results	4	5	1	2	3
Actual results	5	22	2	3	1

This function seems to work great with positive numbers and with no more than two registers. I will continue to experiment and see if specific flags need to be set or cleared and/or if there needs to be another module such as the EXTENDED I/O or HPIL.

RGXTR - Extract Register Address -

Disclaimer: The following discussion is the result of spectulation and experimentation. There was no more information to go on than the name of the function (RGXTR) found in the PANAME ROM. This discussion is incomplete and therefore should not be relied on as accurate.

[RGXTR] (ReGister, eXTRact address) returns the address of the first register of the last line in an array.

Example: You want to know the address of the first register of the last line of an array.
The array pointer 25.04405 is saved in R00. Use the following sequence:

Keystrokes:	Display:	
[RCL] 00	25.04405	Recall the array pointer
[XEQ] "RGXTR"	40.00000	Address of first register of last line

INSTRUCTIONS FOR RGXTR

[RGXTR] replaces, in the X-register, the pointer with the address of the first register of the last line of an array. The old value is saved in LASTx register L.

STACK

Input:	Output:
T: t	T: t
Z: z	Z: z
Y: y	Y: y
X: *bbb.eeeii*	X: address
L: l	L: *bbb.eeeii*

RGSUM - 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, which 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.

column	nº1	nº2	nº3	nº4	nº5	
	R25	R26	R27	R28	R29	
line nº1	-14	15	21	2	8	
	R30	R31	R32	R33	R34	
line nº2	7	13	19	-20	1	
	R35	R36	R37	R38	R39	
line nº3	0	6	12	18	24	ARRAY F
	R40	R41	R42	R43	R44	
line nº4	23	4	5	11	17	
	R45	R46	R47	R48	R49	
line nº5	16	22	3	9	10	

Keystrokes :	Display :	
1 [RCL] 00 [COLPT]	25.04505	1st column pointer.
[RG] "SUM"	32.00000	Sum of elements.
4 [RCL] 00 [COLPT]	28.04805	4th column pointer.
[CHS]	-28.04805	Negative. It specifies a sum of absolute values.
[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, one wants to put in the 3rd column, the percentage of the values of the 4th column, related to its 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 calculate the percentage.
[X<>Y] [RG/Y]          27.04705        End.
```

Now array F is:

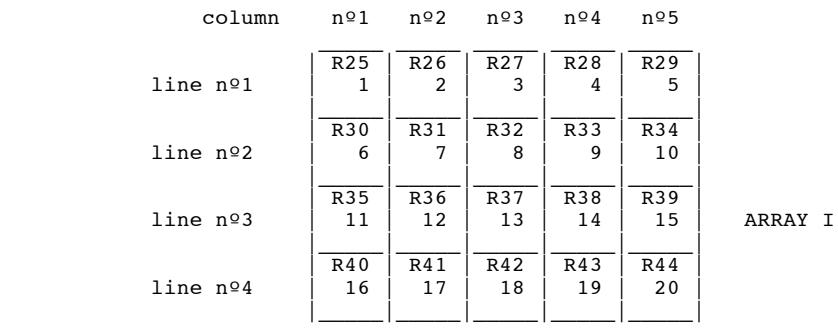
column	nº1	nº2	nº3	nº4	nº5
	R25	R26	R27	R28	R29
line nº1	-14	15	25	2	8
	R30	R31	R32	R33	R34
line nº2	7	13	21.6	-20	1
	R35	R36	R37	R38	R39
line nº3	0	6	10	18	24
	R40	R41	R42	R43	R44
line nº4	23	4	6.6	11	17
	R45	R46	R47	R48	R49
line nº5	16	22	36.6	9	10

ARRAY F'

The 3rd column holds the percentages!

[RGVIEW] (ReGisters VIEW) is a multi-mode function to view, or/and modify registers.

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



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 directions.
[<-]	25.044050	End 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]		Switch off the calculator.
[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
G	26= BCDEFG_	are allowed.
[<-]	26= BCDEF_	It is possible to correct.
[R/S] [BST]	26= BCDEF	Validation and verification.
[SST] A	27= A_	The ALPHA mode is still on.
[ALPHA]	27= 3.000000	Numeric mode.
[EEX] 2	27= 1 2_	
[SST] [BST]	27= 3.000000	Unchanged : no validation with [R/S].
[<-]	-25.044000	Exit.
2 [EEX] 6 [CHS]	2 -6	
[RCL] 00 [+]	25.044052	Array pointer.
[ALPHA] RIEN A	RIEN A_	Array name.
[ALPHA] [RGVIEW]	A1.1= -0.150000	Only the last character is used as
		the array name.
	A1.2= BCDEF	stepping in the array,
	A1.3= 3.000000	is automatic.
[R/S]	A1.4= 4.000000	[R/S] halts it.
[SST] [SST]	A2.1= 6.000000	The element coordinates
[BST]	A1.5= 5.000000	are displayed on the left.
19.5	A1.5= 19.5_	Quick and clear array
[R/S] .	A2.1= ._	input is possible.
[R/S]	A2.2= 7.000000	
[BST]	A2.1= 0.000000	
[<--] 6 [EEX] 6 [CHS]	6 -6_	

```
[RCL] 00 [CHS]            -25.044056         Vector pointer.
[RGVIEW]                  A1= -0.1500000     1st element of the 1st column.
[SST]                     A2= 0.000000       2nd element (R30).
[<--] 3 [RCL] 00 [COLPT]  27.042050          3rd column pointer.
6 [EEX] 6 [CHS] [+]       27.042056
[CHS] [RGVIEW]            A1= 3.000000       1st element (R27).
[ALPHA] LUNDI             A1= LUNDI_
[R/S] MARDI               A2= MARDI_         Input the column,
[R/S] MERCR.              A3= MERCR._        element per element.
[R/S] JEUDI [R/S] [ALPHA] -27.042056         End input and clear ALPHA mode
4 [EEX] 6 [CHS] [ENTER]   0.000004           Creation of a new
3 [RCL] 00 [COLPT] [+]    27.042054          pointer.
[CHS] [RGVIEW]            LUNDI=             In this mode, [RGVIEW]
29                        LUNDI= 29_         allows inputs while the former
[R/S] 12                  MARDI= 12_         value or string
[R/S] [BST]               12.000000=         is still displayed.
[<--] 1 [EEX] 6 [CHS]     1          -6
[RCL] 00 [+] [RGVIEW]     25= -0.150000      In this catalog mode,
                          35= 11.000000      zeros are ignored.
                          40= 16.000000
                          25.044051
```

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.eeeij* pointer.

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

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

When *j* is an odd number, REGisters equal to 0 are ignored.

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 right and 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, and its value.

In ALPHA mode, only the last six characters inputted are kept. 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
```

- OPERATIONS BETWEEN REGISTERS -

RG+- - Addition or subtraction of two vectors -

[RG+-] (ReGisters + or -) adds or subtracts, element to element, two vectors which pointers are in registers X and Y. The sign of the X register specifies the kind of operation.

RG* - Multiplication of two vectors, element to element -

[RG*] (ReGisters *) multiplies the two vectors, element to element, which pointers are in registers X and Y.

RG/ - Divide two vectors, element to element -

[RG/] (ReGisters /) divides, element to element, the two vectors, which pointers are in the X and Y registers.

Example: in the Array below:

- replace the 1st column by the addition, element to element, of the 3rd and 1st column;

- then calculate the square of the 4th column elements;

- finally, divide each of the squares by the 4 first elements of the first line.

The array pointer is saved in register R00.

Array before execution:
Note: Indicated in each square is the register number and its initial value.

column	nº1	nº2	nº3	nº4	nº5	
	R25	R26	R27	R28	R29	
line nº 1	142	20	857	40	1	
	R30	R31	R32	R33	R34	
line nº 2	285	12	714	14	2	
	R35	R36	R37	R38	R39	ARRAY B
line nº 3	428	22	571	24	3	
	R40	R41	R42	R43	R44	
line nº 4	714	32	285	34	4	

```
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 represent 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
```

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

column	nº1	nº2	nº3	nº4	nº5	
	R25	R26	R27	R28	R29	
line nº1	999	20	857	1.60	1	
	R30	R31	R32	R33	R34	
line nº2	999	12	714	9.80	2	
	R35	R36	R37	R38	R39	ARRAY C
line nº3	999	22	571	0.67	3	
	R40	R41	R42	R43	R44	
line nº4	999	32	285	722	4	

INSTRUCTIONS FOR RG+- RG* RG

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

2) The results are saved according to the pointer in the Y-register.

3) After execution, the X-register contains the output vector pointer.

STACK :

```
Input :                    Output :

T: t                       T: t
Z: z                       Z: t
Y: pointer nº1             Y: z
X: pointer nº2             X: pointer nº1

L: l                       L: pointer nº2
```

- SCALAR TO REGISTERS OPERATIONS -

RG+Y

- Add constant to registers -

[RG+Y] (ReGisters + Y) adds the Y-register value to the registers specifies by the X-register.

RG*Y

- Multiply registers by constant -

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

RG/Y

- Divide registers by constant -

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

Example: In Array B below:

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

The array pointer is saved in register R00.

```
                column   nº1   nº2   nº3   nº4   nº5
                        ┌─────┬─────┬─────┬─────┬─────┐
                        │ R25 │ R26 │ R27 │ R28 │ R29 │
        line nº1        │  1  │  2  │  3  │  4  │  5  │
                        ├─────┼─────┼─────┼─────┼─────┤
                        │ R30 │ R31 │ R32 │ R33 │ R34 │
        line nº2        │  6  │  7  │  8  │  9  │ 10  │
                        ├─────┼─────┼─────┼─────┼─────┤
                        │ R35 │ R36 │ R37 │ R38 │ R39 │   ARRAY B
        line nº3        │ 11  │ 12  │ 13  │ 14  │ 15  │
                        ├─────┼─────┼─────┼─────┼─────┤
                        │ R40 │ R41 │ R42 │ R43 │ R44 │
        line nº4        │ 16  │ 17  │ 18  │ 19  │ 20  │
                        └─────┴─────┴─────┴─────┴─────┘
```

```
Keystrokes :        Display :

[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:

	column	nº1	nº2	nº3	nº4	nº5	
		R25	R26	R27	R28	R29	
	line nº1	-4	2	3	4	0.83	
		R30	R31	R32	R33	R34	
	line nº2	1	7	8	9	1.66	
		R35	R36	R37	R38	R39	ARRAY B
	line nº3	12	24	26	28	5	
		R40	R41	R42	R43	R44	
	line nº4	11	17	18	19	3.33	

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 perform on the register value, so results replace initial values.

STACK:

```
Input:                      Output:

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

L: l                        L: l
```

The stack is unchanged by [RG+Y], [RG*Y], [RG/Y].

ALENG - Alpha LENGth -

[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 that 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 lift, if it is enable.

THE STACK

Input:	Output:
T: t	T: z
Z: z	Z: y
Y: y	Y: x
X: x	X: String length.
L: l	L: l

Application program for ALENG.

Example 2: The following routine capitalizes any lowercase letters found in the ALPHA register. It uses [ALENG] to provide a loop counter equal initially 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 codes of leading characters into X.
05  97             The lower case letters are in the range 97 to 122.
06  X>Y?
07  GTO  01
08  CLX
09  122
10  X<Y?
11  GTO 01         If not lower case character, go to [LBL] 01
12  CLX            The character codes for upper case letters
13  32             are determined by subtracting 32 from their
14  -              Lowercase counterparts.
15  R^
16  LBL 01
17  RDN
18  XTOAR          restores capitalized letter to ALPHA.
19  RDN
20  DSE X
21  GTO 00         branches if there is characters remaining.
22  AON
23  .END.
```

ANUM - Search number in ALPHA -

[ANUM] (Alpha to NUMber) searches the ALPHA register, from left to right, for a number. 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 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 unchanged.

2) Numbers in the ALPHA register are considered 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 example 1 string:

Flag 28	Flag 29	Display
set	set	1,234.5000
set	clear	1,0000
clear	set	1,2345
clear	clear	1,2340

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

ANUMDEL -Search ALPHA number and delete -

[ANUMDEL] (Alpha Number DELete) searches the current string in the ALPHA register, form left to right, for a number (represented in numerals) 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" puts this number in the X-register; The ALPHA string is deleted up to "5" included.

INSTRUCTIONS FOR ANUMDEL

1) ANUMDEL searches a numeric value in the string in the ALPHA register. If a number is found, it is put in the X-register and the string is deleted up to the last numerical character of 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 both flag 28 and flag 29 are 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 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.

Flag 28	Flag 29	X-Register	Modified Alpha String
set	set	1,234.5000	XYZ
set	clear	1,0000	,234.5XYZ
clear	set	1,2345	XYZ
clear	clear	1,2340	.5XYZ

STACK

Input: Output:
T: t T: z
Z: z Z: y
Y: y Y: x
X: x X: First numeric value found in ALPHA.

L: l L: l

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	Comments
[SF 28]		Ensures that a comma is not interpreted as a radix.
[CF 29]		Ensures that a comma is not interpreted as a separator of 3 digit groups.
[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"

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

This example shows that "/" and "*" are neither considered as "+", nor as "-", nor as ".".

APPX - Append the integer part of X to ALPHA -

[APPX] (APPend X) appends the integer part of the X-register to the left of the ALPHA register string.

Example: The result of an area calculation 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 integer part 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 has in FIX 0 mode, except that the decimal separator is not appended, and the number is not rounded. As for [ARCL] [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 Rotation -

[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 "ROTA".

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. (i.e. 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.

An operation on a device returns to the ALPHA register the following string "68.2 69.88" (a number, a space, a number). To separately extract two numbers to be used 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.
[STO] 20	68.2000	Store for future use.
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.

POSA - 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 "ABCDEFGHIJ" is in the ALPHA register, what is the position of the 1st "D" character?

```
Keystrokes :            Display :

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 putting 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:

```
Input:                      Output:
T: t                    T: t
Z: z                    Z: z
Y: y                    Y: y
X: code or string       X: position in ALPHA

L: l                    L: code or string
```

SUB$ - Extraction or justification of a sub-string -

[SUB$] (SUB string) extracts a sub-string from the ALPHA register, or it right or left justifies a string, adding spaces to the string.

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

```
Keystrokes :            Display :
2.08                    2.08         2 is the position of "C".
                                     8 is the position of the 7th character to extract.
[XEQ] "SUB$"            2.0800       Extract the string.
[ALPHA]                 CDEFGHI      Sub-string.
```

To right justify in a 10 space field:

```
[ALPHA]
10                      10_          Justification field length.
[CHS]                   -10_         Specifies right justification.
[XEQ] "SUB$"            -10.0000
[ALPHA]                    CDEFGHI   Three space have been inserted to
                                     right justify the string.
```

To put 5 spaces at the right of the string:

```
[ALPHA]                -10.0000
15                      15_          New justification field length.
[XEQ] "SUB$"            15.0000      Left justification because the X-register is
                                     positive.
[ALPHA]                    CDEFGHI
[APPEND]                DEFGHI     _ The ALPHA register scrolls to the left and the
                                     cursor is displayed after the 5 spaces.
```

INSTRUCTIONS FOR SUB$

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

- If the X-register number is an integer, the calculator extracts the |X| left 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 sub-string formed of the characters from the bb position to the ee one (The first character is in position 0). If ee is higher than the position of the last character of the string, the extract string is the initial string, with spaces added to have 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 ee-bb+1 space character string in the ALPHA register.

STACK:

The stack is unmodified by [SUB$].

Note: 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 dots before the string.

- ALPHA AND X TRANSFER FUNCTIONS -

ATOXL - Transfer leftmost character of ALPHA to X -

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

ATOXR - Transfer rightmost character of ALPHA to X -

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

ATOXX - Transfer specified character of ALPHA to X -

[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 unmodified.

XTOAL - Transfer X to leftmost character of ALPHA -

[XTOAL] (X-TO- Alpha Left) Opposite of ATOXL. Inserts a character in the leftmost part of the ALPHA register based on its decimal code in the X-register.

XTOAR - Transfer X to rightmost character of ALPHA -

[XTOAL] (X-TO- Alpha Right) Opposite of ATOXR. Inserts a character in the rightmost part of the ALPHA register based on its decimal code in the X-register.

YTOAX - Transfer character specified in Y to ALPHA at X -

[YTOAX] (Y-TO- Alpha by X) Similar to ATOXX. Returns the character specified by the decimal code in the Y-register to the ALPHA register at the position specified by the X-register. This replaces the character at that position. For $X \geq 0$, the positioning is from left to right, for $X<0$, it's from right to left.

INSTRUCTIONS FOR ATOXL, ATOXR, ATOXX, XTOAL, XTOAR, YTOAX

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 more 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, which position in the string, is specified in the X-register. The ALPHA register is unmodified.

4) [XTOAL] (X-TO- Alpha Left) Opposite of ATOXL. Inserts a character in the leftmost part of the ALPHA register defined by its decimal code in the X-register and shifts the rest of the characters one space to the right. If there are 24 characters in the ALPHA register, including null characters, the rightmost character is dropped.

5) [XTOAR] (X-TO- Alpha Right) Opposite of ATOXR. Inserts a character in the rightmost part of the ALPHA register defined by its decimal code in the X-register and shifts the rest of the characters one space to the left. If there are 24 characters in the ALPHA register, including null characters, the leftmost character is dropped.

6) [YTOAX] (Y-TO- Alpha by X) Similar to ATOXX. Places the character specified by the decimal code in the Y-register into the ALPHA register at the position specified by the X-register. This replaces the character at that position. For $X \geq 0$, the positioning is from left to right, for $X<0$, it's from right to left. The rest of the characters in the ALPHA register are not moved.

A positive value in the X-register specifies a position in the ALPHA register string, starting form the first no null character at the right. This first character is in position 0. This convention is the one used for the POSA function in the XFUNCTION ROM.

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] and [YTOAX] interpretation of the character positions.

Character position	character
n > 23 or r >= string length	DATA ERROR
0 <= n < string length	Nth character after the leftmost
n = 0	Fist character starting from the left
-24 <= n < 0	Nth character starting from the right and up to the register end
n < -24	DATA ERROR

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.

Input:	Display:	
[ALPHA] DECAMETRE [ALPHA]	DECAMETRE	
0 [XEQ] "ATOXX"	68.0000	Code of "D"
4 [XEQ] "ATOXX"	77.0000	Code of "M"
6 [CHS] [XEQ] "ATOXX"	65.0000	Code of "A"
10 [CHS] [XEQ] "ATOXX"	0.0000	Null character

- MEMORY ALLOCATION FUNCTIONS -

PSIZE - Programmable SIZE -

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

SIZE? - Number of data registers -

[SIZE?] returns to the X-register the number of allocated 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.
                  The current number of data registers is in the Y-register.
09 X>Y?           Does the program need more data registers?
10 PSIZE          Change if necessary.
```

READEM - Read all extended memory from mass memory -

[READEM] (READ Extended Memory) copies from a mass memory file (HP82161A tape drive for instance) the whole 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 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
                        TEXTE   A040     by READEM.
                        ....
```

INSTRUCTIONS FOR READEM

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

2) If there is no HP-IL ROM, 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 XMEMORY ROM.

5) If the HP-IL ROM is plug 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.

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

STACK:

The stack is unaffected by [READEM].

INVERSE FUNCTION: WRTEM.

[WRTEM] (WRiTe Extended Memory) copies all the extended memory to a mass medium (HP82161A Tape drive or HP9114).

Example:

```
Keystrokes :        Display :
[XEQ] "EMDIR"       MATRP  P012
                    A      D100   All these files
                    TEXTE  A040   were read with READEM.
                    ....
[ALPHA] "MAT3" [ALPHA]  600.00    Put the file name in ALPHA
[XEQ] "WRTEM"           600.00    All the X-Memory files have been written on the mass
                                  medium.
```

INSTRUCTION FOR WRTEM

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

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

3) If there is a file 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.

/MOD - Euclidean division -

[/MOD] (Divide MOD) calculates the modulo and the quotient of a Euclidean division, that is to say with integers. It is an extension of the [MOD] function of catalog 3.

EXAMPLE

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

Input	Display	
13	13	Dividend input.
[ENTER^] 3	3_	Divisor input.
[XEQ] "/MOD"	1,0000	Modulo.
[X<>Y]	4,0000	Division quotient.
[LASTx]	3,0000	The divisor is saved in the L-register.

INSTRUCTIONS FOR /MOD

1) To calculate the modulo and quotient of the division of Y by X.

2) [XEQ] "/MOD". The quotient and modulo 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 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: Modulo
L: L	L: Divisor

Application programs for /MOD

1) A fairly quick way to calculate 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 subroutine as a small base conversion! This short program, "YBX", gives the digits of the new number; but in reverse. X and Y must be integers.

For instance 1103 [ENTER^] 8 [XEQ] "YBX" returns 7 [R/S] 1 [R/S] 1 [R/S] 2 [R/S] 0. That means: 1103 (DEC) = 2117 (OCT). This result can be checked with the DEC and OCT functions.

N.B : If it is possible to get the divisor back with X<>Y LASTx * + for a quotient >0 and with X<>Y X<0? DSE X NOP * LASTx * + for a quotient <0, it is impossible for a quotient equal to 0.

CHFLAG - Load flag set -

[CHFLAG] (CHarge FLAGs) sets 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 initialize 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 given 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 changes 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?).

```
Ex : 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 and F13, F15 and F16 are used by the printer.

F12		Double width.
F13		Lower case letters.
F15	F16	Printing mode of HP-IL printer.
0	0	Manual
0	1	Normal
1	0	Trace
1	1	Trace and stack printing.

F16
F17 CR-LF ignored
F18
F19
F20
F21 Printing possible
F22 set by a numeric input.
F23 set by an alphanumeric input
F24 Out of range ignored.
F25 Error ignored
F26 Beep on
F27 User mode
F28 Decimal separator type.
F29 Three digit groups separator.
F31 DMY mode of TIME ROM.
F32 MANIO mode on HP-IL ROM.
F34 ADROFF mode on EXTENDED I/O.
F35 Auto start enable (AUTOSTART/DUPLICATION ROM).
F36 to F39 Number of digits for FIX, SCI, ENG.
F40 and F41 Display mode.
F42 and F43 Angular mode.

NOP - No Operation -

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

Example: In a loop one wants to increment the X and Y registers.

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.
```

TF55 - Toggle printer flag -

[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 makes easier the use of some programs (for instance in application pacs), which must be executed with flag 21 set (Printing possible). So you can use them as subroutines, because when flag 55 is cleared, those programs halt at every VIEW or AVIEW instructions. So with [TF55] there is no interruption.

2) Clear flag 55, when a printer is attached to the HP41, which 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 key assignments -

[VKEYS] (View KEYS) with the display of the HP41 lists the key assignments (accessible in USER mode) progressively starting from key R/S, from top to bottom and from right to left. For instance, if the PROMPT function is assigned to the "ENG" key (Yellow key and [3], key code -74), the calculator will display:

```
-74  PROMPT
```

Key listing can be:

- Halted for a while if you press any other key than [R/S] or [ON];

- Ended with the [ON] or [R/S] key. [ON] also switches off the calculator.

Note: [VKEYS] is not programmable.

X<>F - Swap the X-register and flags 0-7 -

The X<>F function swaps the X-register and an 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:

Flag	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

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) Calculate (ie. 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 specified 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 up to 80 new flags. With these extended-flags (0-79) one can:

- 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?.

Consequently, flag 08 of the HP41 reflects the status (set or cleared) of the Nth X-flag.

XSF . XCF and XFS? programs use 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 to 79.

X...NN? - Compare X and a register -

Functions X=NN?. X≠NN?, X<NN?. X<=NN?, X>=NN? and X>NN? work as the normal test functions (ex: X=Y?) of the HP-41, but they work, not between the X-register and other stack registers, but between the X-register and any register specified in the Y-register. These functions are also found in the X-FUNCTION module and are already built into the HP-41CX.

Furthermore these functions also compare alphanumeric strings.

INSTRUCTIONS FOR X...NN?

To compare the X-register and a data register r, put in Y the following:

```
If the r register is:                    Put in Y:

- a data register Rnnn                   - the number nnn
- the Z-register                         - the string 'Z'
                                                    ('Z' ASTO.Y)
- the T-register                         - the string 'T'
- the L-register                         - the string 'L'
```

then compare them.

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

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

These functions compare numeric and alphanumeric functions, following these conditions:

1) A number is always inferior to a string.

2) Strings are compare from the code of their characters
(ex : 'AB0' < 'ABA' because the code of '0' is 48 and the 'A' one is 65.

3) A short string identical to the beginning of a larger one is considered as being inferior. (ex. "ABC" < "ABCD").

Y/N - Question Yes/No -

[Y/N] is useful in programs, which ask for an answer Yes or No.

Example: The following sequence displays the question:

```
END         Y/N?
```

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

```
"END"              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:

message Y/N?

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

1b) To ask another kind of question (for instance: END Y/N) put the message in the ALPHA register, execute AVIEW then Y/N.

2) However, when the Y/N function is executed, the calculator halts program execution and waits for a keystroke:

- If the key hit is the ON key, the calculator is switched off:

- If the key hit is R/S, program execution is stopped and the program pointer is set to the line following Y/N;

- If the key hit is Y (Yes), program execution resumes at the line following directly Y/N;

- If the key hit is N (No), the line following Y/N is ignored, and the program resumes execution at the second line after Y/N (As for a false test; ie. the X=Y? function in the HP41 user manual).

- Any other key is ignored.

Appendix ON

Now, MEMORY LOST is not the only special switch on function! With the PANAME ROM, 6 new functions are possible when you switch on the HP-41.

Note: ON/+ represents the function performed when you hit the [ON] key, while the + key is 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 assignments.

ON/A.

Set the "ALPHA key assignment set", to the top rows. If one key was assigned, its assignment is not modified.

ATOXL	ALENG	ATOXX	ANUMDEL	ATOXR
XTOAL	AROT	YTOAX	ANUM	XTOAR

ON/M.

Like ON/A, but with the following "Matrix (or array) key assignments set".

STO>L	BRKPT	COLPT	AD-LC	RGVIEW
RG	BLDPT	LINPT	LC-AD	CLINC

ON/T.

Like ON/A, but with the following "ploTter key assignments set".

AXIS	BOX	SETORG	RMOVE	*CSIZE
*HOME	RESET	*LABEL	*MOVE	*LDIR
*PLREGX	REVLFX	BACKSPX	RDRAW	*LTYPE
		OUT	*DRAW	COLOR

ON/V.

Like ON/A, but with the following "Video key assignments set".

	SCRLUP	CLEAR	XYTAB	CTYPE
HOME	SCRLDN	CLEARO	CSRL	CSRR
	SCRLX	CSRVX	CSRUP	CSROFF
		CSRHX	CSRDN	CSRON

APPENDIX

The HP-41 uses the "Reverse Polish Notation" (RPN), to solve complex problems, without parenthesis and with a minimum of keystrokes. This system was created by a famous polish mathematician, Lukasiewicz, and not by the Hewlett-Packard company. It is sure that this system demands a few hours of practice for a new user, but it is also sure that its easiness and time saving qualities are worth tenfold these few hours:

- 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 intermediate result, which does not use a register, leaves it free for something else;

- but furthermore the STACK allows for COMPLEX MEMORY MANAGEMENT: one can use a subroutine without modifying it, because of the memory implants the variables in the calling program.

So, subroutine parameters are past throughout the stack, calculations are done in the stack, and results are returned to the stack: a subroutine is general, and easy to use.

In general any arithmetic treatment, 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:

```
c ENTER^ b ENTER^ a XEQ "ROOTS" .
LBL "ROOTS" ST/ Z CHS ST+ X / ENTER^ X^2
RCL Z - SQRT RCL Y SIGN * + ST/ Y .END.
```

For instance to solve the following equation set:

```
 x^2 + x -6 = 0 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: HCD (Higher Common Divisor) of 2 numbers.

```
01 LBL "HCD" LBL 02 MOD LASTx X<>Y X#0? GTO 02 + .END.
91 ENTER^ 65 XEQ "HCD" . X = 13,00
```

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.

The calculation of the HCD is normally done to get the reduced form of a fraction. Also with the HCD of a fraction it is easy to get the SCM (Smallest Common Multiplier or lowest common denominator).

01 LBL "RF" STO Z X<>Y STO T XEQ "HCD" 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 HCD and the SCM of 91 and 65 are Z = 455 and T = 13

Example 4: calculation with two fractions.

The general rule given above is verified here because this operation uses four numbers.

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: Which resistance should be put in parallel with a 100 ohms resistor to get a result of 80 ohms?

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

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