CALCULATOR TIPS & ROUTINES

ESPECIALLY FOR THE HP-41C/41CV

EDITED BY JOHN DEARING

Photograph Courtesy Hewlett-Packard Company

CORVALLIS SOFTWARE, INC.

CALCULATOR TIPS & ROUTINES

ESPECIALLY FOR THE HP-41C/41CV

EDITED BY

John Dearing

CORVALLIS SOFTWARE, INC.
P.O. BOX 1412
CORVALLIS, OREGON 97330

Printed in the United States of America.

P R E F A C E

What is this book? It is a collection of 'tips' and 'routines' for calculators, especially those that use RPN logic, and particularly the HP-41C and HP-41CV. It is a technical reference work to be consulted when needed, rather than a book to be read in detail from cover to cover in one sitting. It is assumed the reader has already studied the operating manuals for his calculator and peripherals. This isn't a 'how-to-program' text, although careful study of these routines may greatly improve your programming ability. A 'tip' is a suggestion or technique for using your calculator more effieiently. Both 'programs' and 'routines' are sets of instructions to a calculator or computer as to what operations are to be performed; a routine cannot be precisely distinguished from a program, but routines are generally short, often-used sets of instructions, frequently called as 'subroutines' by programs. A routine can be thought of as a 'function' that supplements the computing machine's instruction set. It is recommended that the new user of this book read the contents, then leaf through the book to become familiar with what it contains. Later, he should review it more carefully, noting interesting entries.

How was this book made possible? This book is an independent effort, not sponsored by Hewlett-Packard Company or by PPC, the calculator/personal computer users club. However, it would not exist without their help and support. Some of the material in this volume has been used courtesy of H-P; most of it has been contributed by members of PPC. It has been written as a service to all calculator users, to bring together in one volume as many tips and routines as possible, and thus to bridge the gap between operating manuals and books of programs. The numbers after contributors' names are their membership numbers in PPC.

Conventions used: Routine listings are shown as listed by the 82143A Printer, with two exceptions: labels are underlined, not preceded by a diamond; and certain synthetic status registers are shown as they display--see page 108. The first word of the title of every routine using synthetic instructions is 'synthetic'. For more about this topic, read the Foreword, then Chapter XXV. Many readers may wish not to use synthetics, but these instructions and routines using them are too useful to leave out.

How to order copies of this book: Please place all orders in English. The price is $15 postage-paid to North American addresses, US$20 airmail-postage-paid elsewhere. Foreign orders must include a check or money order drawn on a U.S. bank--see your bank or post office. Inquiries by dealers and overseas distributors are invited. Please send all orders and inquiries to: CORVALLIS SOFTWARE, INC., P.O. BOX 1412, CORVALLIS OR 97330, USA.

Contributions are solicited: Submittals of tips and routines for future volumes are requested. Also, should Hewlett-Packard produce a hand-held or portable personal computer, tips and routines for it or them will be wanted. With each contribution, please include a title, an explanation, an example, an instruction listing, your name/address/telephone number, and a signed release statement (on the same sheet, if possible). The release should read: "I, (your name), hereby give permission to Corvallis Software, Inc., to include my tip or routine, (title here), in any book it may publish. They may modify it as they deem necessary. They are under no obligation to use or return this material. If they publish it, they must credit me with my work (unless I desire otherwise), and they must mail me a free copy of the book(s), when published. They are under no further obligation to me." Signed, (your name, both signed and printed or typed).

Pocket Byte (Hex) Table: A miniature Byte Table (see page 112) is available; it is 62 mm by 109 mm (about $2\frac{1}{2}$ by $4\frac{1}{4}$ in), double-sided, with a heavy plastic surface. It fits easily alongside the HP-41 in its case, so that one can always have it with the calculator. The price is $3 for 1, $4 for 2, or $5 for 3. Deduct one dollar from each order if you enclose a self-addressed, stamped envelope. (International orders: pay with a check or money order drawn on a U.S. bank.) Send orders to: KEITH JARETT, DEPARTMENT T&R, 1540 MATHEWS AVENUE, MANHATTAN BEACH, CA 90266 USA.

D E D I C A T I O N

I dedicate this book to the memory of my dear brother,

D O N A L D   A L A N   D E A R I N G

November 23, 1953 — August 16, 1978

For some we loved, the loveliest and the best
That from his Vintage rolling Time hath prest,
    Have drunk their Cup a Round or two before,
And one by one crept silently to rest.

(The Rubaiyat of Omar Khayyam—the 5th edition
of the translation by Edward FitzGerald)

He shall return no more to his house;
Neither shall his place know him anymore.

(Job 7:10)

A C K N O W L E D G M E N T S

I wish to thank the Corvallis Division of the Hewlett-Packard Company for their permission to reproduce material from HP KEY NOTES and other sources. I thank the members of PPC, the independent personal computing users group for enthusiasts and programmers, for the enormous amount of material, ideas and support they have offered. Particular mention must be made of Richard Nelson, who founded PPC (then known as the HP-65 Users Club) in June 1974. Without him, this book would never have been written. Too many members have contributed to this volume to name them all, but the following deserve special mention: Valentin Albillo, Roger Hill, Keith Jarett, John Kennedy, Bill Kolb, Jake Schwartz, Richard Schwartz, and William Wickes. I particularly thank Dr. Wickes for permission to reproduce material from his book, "Synthetic Programming on the HP-41C". Finally, I want to express publicly my appreciation of the love and support of my wife, Peggy, but words fail me: how do you thank someone for having an enduring faith in you?

If you would like to receive a free sample issue of the PPC CALCULATOR JOURNAL, plus membership information, send a self-addressed 9 x 12 inch envelope (it may be folded to fit inside a business envelope) to

PPC CALCULATOR JOURNAL
Department T&R
2545 West Camden Place
Santa Ana CA 92704
USA

For addresses inside the United States, place first class postage for 2 oz on the envelope; for all other addresses, enclose an International Reply Coupon for 2 oz (57 g) of airmail postage. Please do <u>not</u> include a note or letter.

F O R E W O R D

SYNTHETIC PROGRAMMING

New users of the HP-41C/V, and indeed many experienced users, will be surprised upon reading the program listings in this book to encounter a number of 41C program lines that they do not recognize. "STO M" and "RCL b", for example, can not be found in the HP-41C/V owner's manuals; yet they are well defined, quite executable and useful functions. They can be assigned to keys, recorded on cards, etc.--in short, they possess all of the properties of "normal" functions. These new functions are called "synthetic functions", because they are created in the calculator memory by synthesizing together combinations of program bytes that can't be obtained with ordinary keystrokes. A "RCL b" is the result of combining the "RCL" prefix with the "b" postfix (as found normally in "LBL b").

"Synthetic programming" simply refers to any use of synthetic functions in HP-41C/V programming. Stated most concisely, the synthetic program lines constitute an extension of the normal HP-41 function set. Their usefulness depends on the particular application, and on the programmer's creativity--just like any other function. If a programmer doesn't have a use for the "LN" function, he doesn't really care whether it's available. But if he needs it, there's no substitute for it. The same applies to synthetic functions. They perform certain operations--if you can use them, they're great; if you can't, you can forget about them.

The applications of synthetic functions fall into two general categories: program enhancement, and user-machine interaction. For program enhancement, synthetic functions perform certain tasks faster than normal functions, and other tasks that normal functions can't do at all. An example of the latter is the function "RCL d", which recalls a number representing the status of all 56 user and system flags into the X-register. This number can be restored back to its origin at any time via a "STO d" line--thus the user can control the configuration of all HP-41 flags with a single program line or keystroke. The second class of application is in user-machine interaction. An example is synthetic key assignments, where multi-keystroke operations such as GTO IND X (5 keystrokes) can be assigned to a key for single keystroke execution or program entry. The list of examples of applications for synthetic programming is too long for description here--the routines elsewhere in this book serve as prime examples.

The techniques of creating synthetic instructions, which execute on all HP-41C's and 41CV's, have gone through a considerable evolution, primarily through the efforts of members of the PPC. The state-of-the-art at present is represented by the "LB" program (pages 105-113), where synthetic line generation is highly automated, the user having only to enter a series of decimal numbers to identify the byte combinations he desires. Thus the routines in this book can be entered without the user having to understand the principles underlying synthetic programming.

A user wishing to learn the theory of synthetic programming, including details of the purpose and applications of the synthetic functions, is referred to the current and back issues of the PPC Calculator Journal as described in the Acknowledgments and Introduction of this book. A unified description of synthetic programming, including a detailed description of the operating system of the HP-41C/V, is presented in Synthetic Programming on the HP-41C, by William C. Wickes. The book is available from LARKEN PUBLICATIONS, DEPARTMENT T&R, 4517 NW QUEENS AVENUE, CORVALLIS OR 97330, for $10.00 postpaid. (For airmail delivery, include $1.00 additional for the USA, Canada and Mexico, $2.00 for Europe and South America, $3.00 elsewhere.)

William C. Wickes

I N T R O D U C T I O N

The forerunner of the "personal computer" was the HP-65 fully programmable pocket calculator announced in January 1974 by Hewlett-Packard. The HP-65 was the first of a class of machines often described as personal programmable calculators, or PPC's. The HP-65 moved programming from the company computer center into everyones shirt pocket. Today, many "personal computers" that are purchased with the end users funds are designed to be used on a table top connected to the AC power line. All these "personal computers" are inherently limited in speed and memory capacity because of the financial limits of their intended users. An individual can not afford the type of machine used by business and industry. These limits, and the large numbers of users programming these machines, has created the romantic attitude that is accurately described by the user who said, "If I am clever enough I can devise a program to solve almost any problem". The total man hours spent programming personal computers exceeds all man hours spent in programming all computers prior to 1974. With many hundreds of thousands of users writing programs, it is not surprising that a large number of tips and routines have been developed.

The problem with so many programmers developing thousands of tips, techniques, and routines is that there is no practical method of compiling and publishing this material. It almost seems that the more skilled a programmer becomes, the more he or she realizes that there is never enough time to develop all the ideas that come to mind. Because of this, most experienced users of PPC's are eager to add programs and routines to their personal library. Todays hardware is so powerful and physically small that the machine fits into a pocket or small corner of a brief case, but the software fills a filing cabinet.

In the past, the rapid developement of improved models of PPC's discouraged the publishing of books on the subject, because the machine would be out of production shortly after the book reached the market. Most publishers do not want to publish a book dedicated to one machine. If a typical PPC has a life of less than 2½ years, it is almost impossible to produce a book and market it with financial success. It takes a minimum of one year for the user community to master a new machine. Add to that a year to produce the book and you have the two year minimum time required to produce a quality product.

The quality product in this case is a collection of practical tips and routines compiled from all available resources. By not limiting his work to ideas from one individual, John Dearing has been able to draw from the whole user community. This work must be a labor of love and produced using fast and simple production methods, rather than slick paper and colorful graphics.

Many of the tips and routines included in this book have come from PPC members, with permission. This group of users is from the oldest world-wide computer club and they are famous for their super efficient applications of PPC's. One activity of PPC is the discovery and publishing of unsupported features, which has lead to an HP-41C/CV activity known as Synthetic Programming. Thousands of users have tested the routines published in PPC's monthly Journal, the *PPC Calculator Journal*, and there hasn't been a single machine harmed in any way using these unique routines. I am happy to see that John Dearing has succeeded in making this material available to the whole user community.

Richard J. Nelson
Founder of PPC and
Editor of *PPC Calculator Journal*

CONTENTS

NOTICE: "EX" & "MT" (routine 15-19, page 58) appear courtesy David R. Kaplan, 5806 Wood Laurel Court, Burke VA 22015.

CHAPTER I

BASIC FUNCTIONS & OPERATIONS

1-1 TO PUT ANY STANDARD CHARACTER INTO THE X REGISTER: Make sure there are no Alpha characters in the Y Register, and have the "ACCHR" character number (1-127) in X, then XEQ "BLDSPEC". The character will be placed in X, and may be stored in any numeric or stack register, or put into the Alpha Register with "ARCL X". For example, key '40', XEQ "BLDSPEC"; see '(' (left parenthesis) in X. Similarly, '41' gives ')' and '38' gives '&'. See the Byte Table in Reference for others. Many of the characters will display only as a boxed star.

1-2 POSITIONING: In Normal or PRGM mode, 'shift GTO .' followed by a global label positions the 41C/V to that label in program memory (the '.' is not necessary in Normal mode). 'Shift GTO .' followed by a 3-digit number sets the pointer to that line number in the program to which the calculator is currently set (use EEX, then the last 3 digits of the line number, for line numbers 1000-1999). When NOT in PRGM mode, 'shift GTO nn' (where 'nn' is a 2-digit number) sets the pointer to that numeric label in the current program (the next occurance of that numeric label, if it occurs more than once in the current program). Also when not in PRGM mode, 'shift RTN' resets the program pointer to line 00 of the current program.

LOSS OF PENDING RTNs: If any subroutine is executed manually from the keyboard, if 'shift RTN' is pressed, or if SIZE is executed, all pending RTN and END instructions are lost.

DECIMAL POINT & EXPONENTS: In Normal or PRGM mode, if a number consisting of 9 or 10 digits plus an exponent of 10 is to be keyed or entered into a program line, a decimal point must be keyed in before the ninth digit.

R/S VS. STOP: If a program is running, R/S stops the program. Although the STOP function can be assigned to other keys for execution in USER mode, pressing those redefined keys (in Normal or USER mode) will not stop a running program. Furthermore, pressing R/S will stop a running program, even if another function has been assigned to the R/S key location. If a program is not running, pressing R/S starts it running at the current line in the program, unless the R/S key has been reassigned and USER mode is set.

1-3 SYMBOL NAMES: 'Goose': ≯-; 'Backward Goose: -≮; text or super tee: ᵀ; append or lazy tee: ⊦; boxed star or starburst: ▩.

1-4. SLOWING CATALOG REVIEW: Holding any key except R/S slows a catalog review by a factor of seven. Source: Richard Nelson (1) (PPC J, V6N5P32).

1-5 INDIRECT USE OF ALPHA LABELS: Although program labels that are Alpha characters can consist of any of up to seven Alpha characters (except the comma, the period, and the colon), Alpha labels must be held to no more than six characters if they are to be used for indirect addressing. Shorter Alpha labels also save bytes; it might be a good idea to keep them short (six characters at most).

1-6 MISSING CONDITIONALS: For 'X>=0?', use X≠0?, X>0?; for 'X>=Y?', use X≠Y?, X>Y?.

1-7 TO SEE LAST TWO DIGITS IN SCIENTIFIC NOTATION: Key 'EEX nn', where 'nn' is the displayed exponent, including sign, then key '/, FIX 9'. All ten digits are now visible. To recover, key 'LASTX, *'. Source: James Pittman (1002) (65 NOTES, V3 N9P15).

1-8 TO FIT 'TIGHT' HP-67/97 PROGRAMS INTO A BASIC 41C: Try a lower size, then re-size after loading and packing. Source: Richard Nelson (1) (PPC J, V6N5P32).

1-9 MASTER CLEAR: To clear the entire calculator memory, hold down the backarrow (correction) key while you turn the calculator ON. Then release the backarrow key: the "MEMORY LOST" message appears. Clear this message with the backarrow key. If you change your mind at the last instant, release the backarrow key before releasing "ON". Source: Bill Kolb (265).

1-10 BYTES: The basic HP-41C has 63 & 4/7 registers available for program memory. It actually has 64 full registers (448 bytes), but the last 3 bytes contain the permanent .END. statement, leaving 445 bytes for program use. The local Alpha labels A-J and a-e require only 2 bytes, while other Alpha labels need at least 5 bytes (4 + 1 for each Alpha character). If there are k identical bytes repeated N times throughout the program, the number of bytes saved by a subroutine for the k bytes is (N-1)(k-3) - 5, assuming the use of a short-form (00-14) label for the subroutine. Source: HP KEY NOTES, V4N2P11.

1-11 USER KEY AS A PREFIX KEY: The USER key can be used as a true prefix (or shift) key if a keyboard reassignment contains a short routine which ends in 'CF 27', which undoes the USER mode. Example: LBL "SHIFT", FACT, CF 27, RTN. Now assign "SHIFT" to the EEX key. Key an integer from 1 to 69, press 'USER, EEX'. The factorial of the integer will be displayed, and USER mode will be cancelled. Source: Jake Schwartz (1820) (PPC J, V6N7P4).

1-12 ENTERING WITHOUT ENTER: Numbers can be entered into the stack in a program without using the ENTER instruction. After keying one number, switch ALPHA Mode on and off to terminate digit entry (or press ENTER, then BACKARROW keys), then key the next number. This will save paper when listing the program; however, the 41C/V places a nonpackable 'null' after each line of numeric entry followed by another line of numeric entry. This null takes one byte; therefore this technique will not save bytes. Source: Richard Nelson (1) (PPC CJ, V7N2P56).

1-13 LOCK OFF ("LO"): LBL "LO", LBL 00, SF 11, OFF, GTO 00, END. When ON is pressed, BEEP sounds and the 41C/V immediately turns off, preventing those unfamiliar with its operation from tampering with it. This routine also helps avoid running down the batteries when carrying the calculator. To override "LO", hold down the R/S key while pressing the ON key. Source: HP KEY NOTES, V4N1P4; Bill Kolb (265).

1-14 SIZE FINDER ("SZE"): This routine will find the current SIZE (the number of data registers assigned). Source: Richard Nelson (1) (PPC CJ, V7N1P0).

01 LBL "SZE"	08 ENTER	15 FRC	22 X<>Y	29 GTO 01	36 ARCL X
02 0	09 .32064	16 X=Y?	23 4	30 LBL 09	37 AVIEW
03 SF 25	10 LBL 01	17 GTO 09	24 /	31 LASTX	38 FIX 2
04 ARCL IND X	11 ISG X	18 LASTX	25 .24	32 INT	39 END
05 FC? 25	12 ARCL IND X	19 DSE X	26 +	33 LBL 10	
06 GTO 10	13 FS? 25	20 ABS	27 +	34 FIX 0	
07 .32001	14 GTO 01	21 INT	28 SF 25	35 "SIZE= "	(80 bytes)

1-15 FAST SIZE FINDER ("SZ"): Source: Ron Knapp (618) (PPC CJ, V7N2P38).

```
01 LBL "SZ"
02 9
03 ENTER
04 -1
05 ENTER
06 512
07 LBL 01
08 2
09 /
10 SF 25
11 ST+ Y
12 ABS
13 ARCL IND Y
14 FC?C 25
15 CHS
16 DSE Z
17 GTO 01
18 X<0?
19 CLX
20 +
21 FIX 0
22 "SIZE= "
23 ARCL X
24 AVIEW
25 FIX 2
26 END
```

(51 bytes)

1-16 SYNTHETIC SIZE FINDER ("?S"): This routine returns the current SIZE to the X Register. The old value in X is saved in Y, Z & T, while 'garbage' is left in Alpha. WARNING: As with many programs that use the results from the flag register, a wrong final answer will be obtained if the routine is single-stepped between lines 6 and 9 (because Flag 51 will be set), or even if the routine is stopped and immediately restarted between those lines while the printer is connected (because Flag 55 will be set). Line 04 is append 4 nulls. Source: Roger Hill (4940) (PPC CJ, V7N5P57).

```
01 LBL "?S"
02 RCL c
03 STO M
04 "├◆◆◆◆"
05 X<> M
06 X<> d
07 CF 02
08 CF 03
09 X<> d
10 ENTER
11 INT
12 HMS
13 7
14 *
15 +
16 -10
17 *
18 INT
19 64
20 MOD
21 SF 25
22 LBL 00
23 RCL IND X
24 FC? 25
25 RTN
26 X<> L
27 +
28 GTO 00
29 END
```

(56 bytes)

1-17 SYNTHETIC SIGMA, SIZE & CURTAIN FINDER ("Σ?", "S?" & "C?"): XEQ "Σ?" to find the number of the first register of the statistics block. XEQ "S?" to find the SIZE. XEQ "C?" to find the curtain location (the absolute address of data register 00). Line 26 is decimal 244, 127, 0, 0, 1. Source: Keith Jarett (4360) & Roger Hill (4940) (PPC CJ, V7N10P15; PPC ROM).

```
01 LBL "Σ?"
02 CLA
03 XEQ "C?"
04 RCL N
05 XEQ 14
06 X<>Y
07 -
08 RTN
09 LBL "S?"
10 XEQ "C?"
11 CHS
12 64
13 MOD
14 SF 25
15 LBL 02
16 RCL IND X
17 FC? 25
18 RTN
19 X<> L
20 +
21 GTO 02
22 LBL "C?"
23 RCL c
24 LBL 14
25 STO M
26 "├◆◆x"
27 X<> M
28 X<> d
29 CF 01
30 CF 02
31 CF 04
32 CF 07
33 FS?C 10
34 SF 07
35 FS?C 11
36 SF 09
37 FS?C 12
38 SF 10
39 FS?C 13
40 SF 11
41 FS?C 14
42 SF 13
43 FS?C 15
44 SF 14
45 FS?C 16
46 SF 15
47 X<> d
48 E2
49 *
50 INT
51 DEC
52 END
```

(112 bytes)

1-18 SYNTHETIC SUSPEND & REACTIVATE KEY ASSIGNMENTS ("SK" & "RK"): To suspend all system and program key assignments, key in a register pointer, 'n', then XEQ "SK"; key assignments will be stored in R'n' and R'n+1'. To reactivate these key assignments, key 'n', XEQ "RK". Minimum SIZE is n+2. Values in X, Y & Z before keying 'n' are restored. Step 24 is nonstandard; it is decimal 243, 127, 15, 255. Source: Keith Jarett (4360) (PPC ROM).

```
01 LBL "SK"
02 SIGN
03 CLX
04 X<> ├
05 XEQ 14
06 ISG L
07 ""
08 .
09 X<> e
10 LBL 14
11 "*"
12 X<> M
13 STO N
14 ASTO IND L
15 RDN
16 RTN
17 LBL "RK"
18 SIGN
19 ARCL IND L
20 "├◆"
21 ISG L
22 ""
23 ARCL IND L
*24 "├Φ"
25 X<> N
26 STO ├
27 X<> M
28 STO e
29 RDN
30 CLA
31 END
```

(64 bytes)

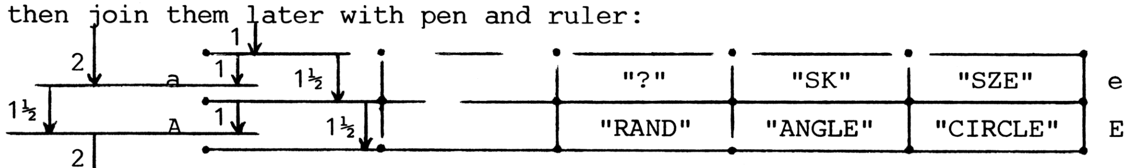
1-19 MARKING OVERLAYS: Press-on letters are a viable alternative to HP adhesive labels for marking keyboard overlays; use a clear matte spray to protect letters once they are applied.

1-20 KEYING EXPONENTS: It is not necessary to enter two digits in an exponent that has just one. Source: Bill Kolb (265).

1-21 SYNTHETIC VIEW KEY ASSIGNMENTS ("VK"): This routine determines which keys are reassigned (to either system functions or user programs). CASE I: NO PRINTER PLUGGED IN: routine pauses to display assignments by keycode, then goes to the next key in numeric order (11 to 84). CASE II: PRINTER PLUGGED IN: (A) Printer ON: "PRKEYS" executed. (B) Printer OFF: "PRINTER OFF" displayed -- do any of the following: (1) turn printer ON, reexecute "VK"; or (2) unplug printer, reexecute "VK"; or (3) switch to PRGM Mode, then do either (a) or (b): (a) SST 3 times, switch out of PRGM Mode, R/S. Behaves as if no printer is in system (Case I); clears Flag 21. (b) SST 4 times, switch out of PRGM Mode, R/S; routine stops to display assignment(s) to a given key; R/S for next key. NOTE: Lines 11 and 16 are nonstandard. Line 11 is decimal 243, 127, 16, 240; line 16 is decimal 243, 127, 32, 240. Source: Roger Hill (4940) & Tom Cadwallader (3502) (PPC ROM).

```
 01 LBL "VK"
 02 SF 21
 03 FS? 55
 04 PRKEYS
 05 FS? 55
 06 RTN
 07 CF 21
 08 8
 09 RCL ├
 10 XEQ 07
*11 "├θ"
 12 X<> M
 13 X<> d
 14 RCL e
 15 XEQ 07
*16 "├ "
 17 X<> Z
 18 X<> M
 19 LBL 01
 20 -27.00008
 21 RCL M
 22 -
 23 STO N
 24 RDN
 25 LBL 02
 26 FC? IND N
 27 FC? 50
 28 GTO 05
 29 X<> d
 30 FC? IND N
 31 FC? 50
 32 GTO 05
 33 X<> d
 34 LBL 03
 35 ISG N
 36 GTO 02
 37 DSE M
 38 GTO 01
 39 X<>Y
 40 LBL 04
 41 STO d
 42 CLST
 43 CLA
 44 PSE
 45 CLD
 46 RTN
 47 LBL 05
 48 X<> d
 49 35
 50 RCL N
 51 INT
 52 +
 53 OCT
 54 1
 55 ST+ Y
 56 %
 57 +
 58 10
 59 MOD
 60 LASTX
 61 *
 62 INT
 63 STO a
 64 43
 65 -
 66 ABS
 67 1
 68 X<Y?
 69 ST+ a
 70 FS? 42
 71 FC? IND N
 72 CHS
 73 ABS
 74 X<> M
 75 RDN
 76 X<> N
 77 RDN
 78 " "
 79 FC? 42
 80 "   -"
 81 FS? 50
 82 X<> d
 83 X<>Y
 84 FC? 50
 85 GTO 04
 86 X<> d
 87 X<> Q
 88 CLX
 89 RCL d
 90 FIX 0
 91 CF 29
 92 ARCL a
 93 ISG L
 94 GTO 06
 95 "├ -"
 96 ARCL a
 97 LBL 06
 98 STO d
 99 X<> Q
100 AVIEW
101 TONE 0
102 R↑
103 STO N
104 R↑
105 STO M
106 R↑
107 R↑
108 X<> d
109 X<>Y
110 FS? 42
111 X<> d
112 GTO 03
113 LBL 07
114 CLA
115 X<> M
116 "├*****"
117 X<> N
118 X<> M
119 END
    (223 bytes)
```

1-22 TYPING TOP ROW KEY ASSIGNMENTS: Frequently, only the keys in the top two rows are assigned (often with the local labels A-J and a-e). If the documentation of a program is being typed, here is a way to represent these assignments. Type the dots, then join them later with pen and ruler:



To center a typed label in a given 'box', set typewriter to first space in box, then space once for each character in the label. Next, space to the dot at the right of the box, counting the spaces. Divide this number by 2, then backspace to the first space in the box and space in this number of spaces; type the label. To indicate key assignments anywhere on the keyboard, draw an outline of a keyboard overlay, then type or write in the assignments -- shifted assignments on the keys, unshifted above. Source: John Dearing (2791).

1-23 CRASH: A non-responsive display state during which some or all of the keys are locked out (won't function). Causes of crashes include shock to the calculator and misuse of some synthetic functions. Usually, removing and then immediately reinserting the battery pack will cause recovery without loss of memory. In extreme cases, it may be necessary to leave the batteries out over night or for as long as 48 hours. Another extreme case solution is suggested by Jim DeArras: start a card through the reader and remove the batteries with the card half through. Leave the reader in the calculator 2 minutes, then replace the batteries. A "MEMORY LOST" should result. Source: Bill Kolb (265).

1-24 FUNCTION PRIORITY OF THE TOP ROW KEYS: When the 41C/V is in USER mode and you press A-J (any key in the top two rows) or a-e (SHIFT, then any key in the top row), the calculator does the following:

1. If the key has been reassigned, it performs the reassigned function.
2. Else, it searches for the corresponding local Alpha label (A-J, a-e) in the current program only, and executes it if found.
3. Else, it executes the printed Normal Mode function.

Because of this, execution of a Normal Mode function on a key in the upper two rows can be rather slow, when in USER Mode. To shorten this time, press 'SHIFT GTO ..', turn USER off, or assign the Normal Mode function to its own key.

1-25 SIGN: The "SIGN" function is not a true unary function. The unary function should return zero rather than 1.00 when the argument is zero. To obtain the unary, substitute the following lines, as suggested by Leland Van Allen: 'STO L, X≠0?, SIGN'. Source: Bill Kolb (265).

1-26 SYNTHETIC KEY ASSIGNMENTS CLEAR ("KC"): XEQ "KC" to clear all key assignments, both system and user. Line 04 is nonstandard; it is decimal 246, 127, 1, 105, 11, 240, 0. Source: Paul Lind (6157). See 21-8.

```
01 LBL "KC"
02 RCL c
03 STO M
*04 "├×iλ◆"
05 RCL M
06 X<> d
07 15.006
08 ENTER
09 .
10 ENTER
11  E
12 LBL 01
13 FS? IND Z
14 ST+ Y
15 ST+ X
16 DSE Z
17 GTO 01
18 RDN
19 192
20 -
21 .
22 R↑
23 X<> d
24 X<> c
25 X<>Y
26 LBL 02
27 STO IND Z
28 DSE Z
29 GTO 02
30 STO e
31 STO ⊤
32 X<>Y
33 STO c
34 CLST
35 END
(72 bytes)
```

1-27 SYNTHETIC BYTES SAVED WITH A SUBROUTINE ("BS"): This routine calculates the bytes saved (or used) by using a subroutine for repeated keystrokes. Load the following in the stack before executing "BS":

Z: R--# of repeated bytes, not including the LBL or RTN of the proposed subroutine. Make the number negative if indirect calls are being made.
Y: C--# of calls made to the subroutine. Make negative if a short-form local label (labels 00-14) is being called.
X: A--# of Alpha characters in the GLOBAL label. Use zero if the label is a local label (LBL 01, LBL 25, LBL A, LBL e, etc).

To use, key R, ENTER, C, ENTER, A, XEQ "BS". The output is the number of bytes saved. If the number is negative, the proposed subroutine takes more bytes than if the sequence is repeated in the program. For indirect calls, the routine counts the 7 bytes required for the indirect register also; if using a register that is available anyway, add 7 to the output. Source: Charles Close (3878) (PPC CJ, V8N1P14).

BYTES:

XEQ numeric: 3
XEQ indirect: 2
XEQ Alpha: 2 + 1 per char.

LBL 00-14: 1
LBL 15-99: 2
Local α labels: 2
Global α labels: 4 + 1 per character
RTN: 1

```
01 LBL "BS"
02 0
03 X<> d
04 RDN
05 X≠0?
06 SF 03
07 STO M
08 RDN
09 X<0?
10 SF 04
11 ABS
12 STO N
13 X<>Y
14 X<0?
15 SF 01
16 ABS
17 STO O
18 *
19 FC? 01
20 SF 02
21 FS? 03
22 CF 02
23 X<>Y
24 X<> N
25 2
26 FS? 02
27 1
28 FS? 02
29 +
30 X<>Y
31 *
32 LASTX
33 RCL M
34 *
35 FC? 01
36 +
37 FS? 01
38 RDN
39 -
40 RCL M
41 -
42 RCL O
43 -
44 2
45 FS? 04
46 GTO 01
47 FC? 03
48 1
49 FS? 03
50 3
51 +
52 LBL 01
53 FS? 01
54 7
55 FS? 01
56 +
57 -
58 RCL N
59 STO d
60 RDN
61 CLA
62 END
(96 bytes)
```

CHAPTER II

PROGRAMMING TIPS

2-1 TO GO TO/DELETE TO THE END OF A PROGRAM, FROM ANY LINE THEREIN: (a) to go to the END: for programs of up to 999 lines, press 'SHIFT GTO .999'. For programs of up to 1999 lines, press 'SHIFT GTO . EEX 999'. (b) to delete the rest of a program, beginning at the current line: execute 'DEL 999'; this works for up to 999 lines remaining; the END statement will not be deleted. Note that you do not recover the deleted lines (bytes) until after packing. Source: Bill Kolb (265).

2-2 PROGRAMMING CHANGES: When you must make program changes using existing program line numbers as references, you should change the last line (or group of lines, using "DEL") first. In this way you move 'up' (to lower line numbers) through the program as you make changes, and steps remaining to be changed retain their original line numbers. To change a step, delete first, then insert. Source: Richard Nelson (1) (PPC J, V5N2P9).

2-3 "BST" AND EDITING: For editing convenience, assign "BST" to another unshifted key, such as "TAN". Source: Richard Nelson (1), (PPC J, V6N5P32).

2-4 EFFECT ON X- AND Y-REGISTERS OF SOME FUNCTIONS: With these equations in mind, these functions can often be used as a shortcut in a program. For example, to calculate the square root of the sum of the squares of the values in the X- and Y-Registers, just press "R-P". Postscript 1 indicates the contents of the given register before execution of the function, and postscript 2 indicates the contents of the given register after execution. L = LastX Register. Source: 'HP-41C Owner's Handbook and Programming Guide', © Copyright (January 1979) Hewlett-Packard Company. Reproduced with permission.

P - R (Y1 lost)

$Y2 = X1 \sin Y1$
$X2 = X1 \cos Y1$
$L2 = X1$

R - P (Y1 lost)

$Y2 = \arctan (Y1/X1)$
$X2 = \sqrt{X1^2 + Y1^2}$
$L2 = X1$

D - R

$Y2 = Y1$
$X2 = \frac{X1\ pi}{180}$
$L2 = X1$

R - D

$Y2 = Y1$
$X2 = \frac{X1\ 180}{pi}$
$L2 = X1$

%

$Y2 = Y1$
$X2 = \frac{X1\ Y1}{100}$
$L2 = X1$

%CH

$Y2 = Y1$
$X2 = [(X1 - Y1)\ 100]/Y1$
$L2 = X1$

SIGN

$Y2 = Y1$
$X2 = -1, 0 \text{ or } 1$ **
$L2 = X1$

** SIGN returns -1 if X was negative, 0 if X contained Alpha characters, and 1 if X was zero or positive. See 1-25.

2-5 REEXECUTING THE CURRENT PROGRAM: You can press 'XEQ, ALPHA, (name of program), ALPHA'; you can also assign the program to a key, then press that key in USER Mode. Here's another way: if you know the program stops execution at the END (last line) of the program, just press R/S. If execution stops at an internal STOP or RTN, press 'SHIFT RTN R/S'.

2-6 VARIABLE LENGTH "PAUSE" WITHOUT BLINKS ("VP"): Not a real "PSE" -- the program is still running. The number you put in Line 02 determines the length of the pause (n=10 gives about a 1 second pause; 100, 10 seconds). The original contents of the T Register is replaced by zero. Execute as a subroutine, or insert steps 02-08 into a program. For more flexibility, change step 02 to RCL 00, and store 'n' in R00 before execution. Source: Tom Cadwallader (3502) (PPC J, V6N6P21).

```
01 LBL "VP"     03 RDN        05 LBL 14    07 GTO 14    09 END
02 (n here)     04 VIEW X     06 DSE T     08 CLD
```

After a program executes an AVIEW or VIEW, the flying goose will not appear, but the program annunciator will be displayed; any time a program places an Alpha string into the display, that string replaces the goose program execution symbol. When the program clears the display, or the program is interrupted, the symbol returns to the display. Source: 'HP-41C Owner's Handbook and Programming Guide', © (Jan 1979) H-P.

2-7 TEST DETERMINES FUNCTION: 'X(?)Y' stands for any conditional test, such as "X=Y?". (1) Add or subtract, depending on test: 'X(?)Y, CHS, +'. Subtracts if conditional is true, adds if false. (2) Multiply or divide, depending on test: 'X(?)Y, 1/X, *'. Divides if conditional is true, multiplies if false. (3) Power or root, depending on test: 'X(?)Y, 1/X, Y↑X'. Takes root if conditional is true, takes power if false. Source: Bill Kolb (265).

2-8 DO TWO STEPS IF CONDITIONAL IS TRUE; SKIP IF FALSE: Use the same conditional twice, following each with a step to be executed if test is true. For example, 'X=Y?, PSE, X=Y?, GTO 08' will pause and then go to LBL 08 only if X=Y; if X≠Y, execution will branch around both of these instructions.

2-9 LAST "RTN" OR "END" NOT NEEDED: A program loaded at the end of program memory (i.e., a program keyed in after 'SHIFT GTO ..' is pressed) need not be terminated with an END or RTN -- the permanent .END. will serve. Source; 'HP-41C Operating Manual', © Copyright (June 1980) Hewlett-Packard Company. Reproduced with permission.

2-10 EXECUTING A NUMERIC-LABELED ROUTINE IN ANOTHER PROGRAM: Many independent subroutines in a program can be headed by a single global label, such as "MIS" (miscellaneous), and each subroutine can be headed by a numeric label. The user can call any of the subroutines by keying in its numeric label (manually or under program control), followed by XEQ "MIS". Rolling the index 'down' to the T Register gets it out of the way of useful data. Source: William Cheeseman (4381) (PPC CJ, V7 N5P9).

LBL "MIS", RDN, GTO IND T, LBL 00, ..., RTN, LBL 01, ..., RTN, ..., END.

2-11 INDIRECT XEQ OR GTO WON'T WORK FOR LOCAL ALPHA LABELS (A-J, a-e): Example: a large letter banner-printing program that uses a printing routine for each letter. If an 'A' is to be printed, it is most convenient to store the 'A' in the X Register and XEQ IND X. But LBL A (through LBL J) won't execute indirectly. LBL "AA" (and all other such 'double-character' labels) will, however, as they are Global Labels. It is an easy matter to make all labels double labels and double the letters as they come up. The Alpha register is used for this process. The program instructions would be:

```
ASTO X     Stores "A" in the X Register.
ARCL X     Adds "A" to Alpha, giving "AA".
ASTO X     Stores "AA" in the X Register.
XEQ IND X  Executes Global Label "AA".
```

The use of indirect addressing saves so much program memory that the added byte for each label to make a double letter label is still memory efficient. Source: HP KEY NOTES, V4N1P11. Note: synthetic Global Labels A - J can also be used.

2-12 DUPLICATING FUNCTION AND PROGRAM NAMES: A function and a program both having the same name can be executed from the keyboard if the function is assigned to a key before the program label by the same name is keyed into memory. The program can then be assigned to another key or executed manually. The same thing can be done with two or more programs having duplicate names, by simply assigning the labels to separate keys as each label is keyed into memory. The 41C/V keeps the assignment straight. Source: HP KEY NOTES, V4N2P11.

2-13 RTN TO END: A program can be divided into two programs by changing an internal "RTN" to an "END". Go to the RTN statement in PRGM mode, delete the step, and then press 'XEQ, ALPHA, E, N, D, ALPHA' ('GTO ..' won't work). Be sure you have a global label after the RTN first, if you want the program following the RTN to start with a global label. Now you can record a portion of a program, using the card reader, after isolating it with ENDs. This program segment can be placed elsewhere, and then the preceeding END deleted to combine the new segment. Source: John Dearing (2791) (PPC CJ, V8N1P14).

2-14 EXECUTING A SERIES OF STEPS WITHIN THE MAIN BODY OF A PROGRAM MORE THAN ONCE: (Rather than as a subroutine). Both examples execute the routine twice:

Case I: Using a loop control: Use ISG or DSE to control the number of loops. Ex.: ..., 2, STO 00, LBL 01, ... (routine here) ..., DSE 00, GTO 01, ...

Case II: At the end of the program: (Two XEQs would run the routine 3 times). Ex.: ..., XEQ 02, LBL 02, ... (routine here) ..., END. Source: John Dearing (2791).

2-15 NO OPERATION ("NOP"): A NOP is a step that does nothing (or nothing harmful); used after "ISG", it effectively changes the ISG to a simple increment instruction (no skipping); similarly, placed after a "DSE", a NOP changes it to a decrement instruction (again, no skipping). Depending on the situation, you can use DEG, RAD, GRAD, FIX, SCI, ENG, CF any, SF any, or LBL any. A LBL instruction is the fastest nonsynthetic instruction, and has the advantage of being a one-byte NOP if numbered less than 15. Two instructions that can be used at any time are STO X and X<>X. For synthetic programmers, the best NOP is "" (Text 0 -- byte 240).

2-16 ITERATION OR LOOP COUNTER: To count the number of times a loop is executed, include an "ISG nn" instruction in the loop (where 'nn' specifies a register that contains zero initially), and follow the ISG step with any NOP such as STO X. When execution stops, the loop count is obtained by recalling Register 'nn'. Source: Richard Nelson (1) (65 NOTES, V1N2P3).

2-17 LOCAL LABELS: When a global label is accessed within a program with GTO or XEQ, execution speed can be increased by placing a numeric label after the global label (or replacing the global label with a numeric label), and changing the GTO or XEQ instruction(s) so they refer to the numeric label. Numeric label search is much faster than global label search. Short-form labels (LBLs 00-14) should be within 112 bytes of a GTO instruction so the calculator can remember the label's location. If the LBL is more than 112 bytes from its GTO instruction, use LBLs 15-99. Since local Alpha labels (A-J, a-e) cannot be used indirectly with GTO or XEQ, put a numeric label after them (example: LBL A, LBL 01) if they need to be executed indirectly, and use the numeric label for the indirect reference. "XEQ" instructions do not need to be within 112 bytes of LBLs 00-14.

2-18 USE 'RDN, RCL' RATHER THAN 'CLX, RCL': This avoids the problem of stack lift enable when you manually stop execution immediately before the RCL instruction. Source: Keith Jarett (4360) (PPC CJ, V7N8P9).

2-19 POWER FAILURE PROTECTION DURING LONG PROGRAM RUNS: The sequence 'FS? 49, OFF' may be used to protect against power failure. Source: Richard Nelson (1).

2-20 R↑ OR RDN?: If possible, use R↑ to save time: RDN takes 17.4 ms; R↑ takes 12.8 ms. Thus two R↑s are better than two RDNs. Often stack manipulations can be rewritten to favor use of R↑ over RDN. Source: Richard Nelson (1) (PPC CJ, V7N8P8).

2-21 TO BE ABLE TO RERUN A PROGRAM WITH "R/S": If execution stops at the last step, just press R/S to rerun the program. To be able to rerun a program with R/S when execution stops at an internal RTN or STOP, follow the RTN or STOP with a GTO instruction that points to the first step of the program.

2-22 TO MAKE AN INTERNAL STOP 'FINAL': When execution stops inside a program, and you want to prevent R/S from inadvertantly executing the following portion of the program, then, instead of just stopping with "STOP" (or "RTN"), use LBL 14, STOP (or RTN), GTO 14.

2-23 ADD (OR SUBTRACT) A GIVEN VALUE ONLY IF CONDITIONAL IS FALSE: Follow given value with any conditional ['X(?)Y'], then with "CLX", then "+" (or "-"). For example, to add 5 to the value in X only if Flag 00 is clear, use '5, FS? 00, CLX, +'.

2-24 TURNING OFF WHILE IN PRGM MODE: If you turn off the 41C/V (or if it turns off automatically) while it is in PRGM mode, you should toggle into and back out of PRGM mode when you resume operations. This ensures that changes made to programs in previous editing sessions will be compiled by the calculator. Source: HP KEY NOTES, V4N1P3.

2-25 ROUTINE MESSAGE: For a long-running routine, put a message in Alpha (12 characters or fewer -- like "SORTING"), followed by AVIEW; at the end of the routine, put CLD. This will tell you what the program is doing.

2-26 FIVE SECOND PAUSE: When the card reader is plugged in but the printer is not, use "7PRTX" for a long pause. Source: PPC Melbourne Chapter.

2-27 TO INSERT PROGRAM LINES AHEAD OF STEP 01: Press 'GTO . 000', then enter the desired steps. If in Run (Normal or USER) Mode, use 'SHIFT RTN', then switch to PRGM Mode and enter desired steps. Source: Bill Kolb (265).

2-28 ALPHA STRING AS INDIRECT ADDRESS: An indirect address can be an Alpha string as well as a number. This feature can be used to create a directory or can be used in word games. Source: Bill Kolb (265).

2-29 UNLABELED PROGRAMS: If you accidentally delete a program label or if you have an unlabeled program in memory, you can find it again using 'CAT 1'. XEQ CAT 1, press R/S as soon as the first program name appears, then use SST and BST to find the second of two consecutive END statements (no label in between). Switch to PRGM mode and delete the program, or press 'SHIFT GTO . 000', then key a global label for the program. It's a good idea to check for unlabeled programs when you are running low on memory. Source: Bill Kolb (265).

2-30 NO END: Don't put an END on a program until you wish to add another, autonomous program. Programming an END puts the program pointer into a new region, making return to the initial program unnecessarily complicated. Don't use 'GTO ..' to find out how many registers are left: use 'GTO . 000' or 'GTO . nnn', where 'nnn' is larger than the number of lines in the program. Source: Bill Wickes (3735).

2-31 PRINT ALPHA IF POSSIBLE, BUT AVOID SCROLLING: Instead of using AVIEW to print the contents of the Alpha Register, use SF 25, PRA, CF 25 to avoid scrolling a long Alpha string across the display, when the Alpha message doesn't need to be seen when printing can't occur. Source: John Herzfeld (5428). See 3-12, 4-1, 6-14, 22-23.

2-32 SYNTHETIC SHORT-FORM GTO WITH FULL DISTANCE MEMORY: Conventional wisdom is that you use short-form LBLs and GTOs if all the GTOs are within 112 bytes of the LBL: otherwise you use the 15 and up variety (see routine 2-17). Paul Lind (6157) has noticed that one can create GTOs 00-14 with three bytes and full-distance memory. Even just one short-form LBL and a 3-byte GTO will save a byte over a long-form LBL and GTO. The savings is greatly increased if a LBL is called by several GTOs, some within 112 bytes, some beyond. Only the GTOs beyond 112 bytes need to be of the 3-byte variety. A 3-byte GTO nn is easy to make with the Synthetic Load Bytes program "LB" (Chapter 25); the input is decimal 208, 0, 0-14 (for example, a 3-byte GTO 00 is 208, 0, 0; a 3-byte GTO 13 is 208, 0, 13). The second byte can be anything. Source: John Herzfeld (5428).

2-33 SYNTHETIC LENGTHEN & SHORTEN RETURN STACK ("LR" & "SR"): Six return pointers are stored in Rx and R(x+1) when "LR" is executed; when you are five levels deep in subroutines and need to lengthen the return stack, enter register number, XEQ "LR". When you are returning from more than six levels of subroutines and have used "LR", then key in the register number and XEQ "SR" to place the next 6 levels of return addresses in the status registers. Mark subroutine levels in groups of 5. If execution is to go more than 1 subroutine level beyond any of these dividing lines, the last subroutine level of the previous group must execute "LR" & "SR". See example below. In this example, if a RTN were placed after LBL 06 (Line 30), then Lines 23-4 & 26-7 (executing "LR" & "SR") were deleted, execution would still stop after the first RTN (Line 05). If the RTN were placed just after LBL 07 (Line 34) instead, however (still with Lines 23-4 & 26-7 deleted), execution of "X" would stop after the second RTN (Line 09). If Lines 23-4 & 26-7 are restored at this point, all RTNS would again be executed. As written, the example will pause to view "LBL 16" (to demonstrate that execution did go 16 subroutine levels deep) (R/S if printer is plugged in but off); then execution resumes until the '1' of Line 04 is displayed (demonstrating that execution returned all the way back to Line 05 [RTN]). Source: PPC ROM. Harry Bertuccelli (3994).

Example Listing:

```
01 LBL "X"
02 CLX
03 XEQ 01
04 1
05 RTN
----------
06 LBL 01
07 XEQ 02
08 2
09 RTN
10 LBL 02
11 XEQ 03
12 3
13 RTN
14 LBL 03
15 XEQ 04
16 4
17 RTN
18 LBL 04
19 XEQ 05
20 5
21 RTN
22 LBL 05
23 0
24 XEQ "LR"
25 XEQ 06
26 0
27 XEQ "SR"
28 6
29 RTN
-----------
30 LBL 06
31 XEQ 07
32 7
33 RTN
34 LBL 07
35 XEQ 08
36 8
37 RTN
38 LBL 08
39 XEQ 09
40 9
41 RTN
42 LBL 09
43 XEQ 10
44 10
45 RTN
46 LBL 10
47 2
48 XEQ "LR"
49 XEQ 11
50 2
51 XEQ "SR"
52 11
53 RTN
---------
54 LBL 11
55 XEQ 12
56 12
57 RTN
58 LBL 12
59 XEQ 13
60 13
61 RTN
62 LBL 13
63 XEQ 14
64 14
65 RTN
66 LBL 14
67 XEQ 15
68 15
69 RTN
70 LBL 15
71 XEQ 16
72 16
73 RTN
74 LBL 16
75 "LBL 16"
76 AVIEW
77 PSE
78 CLD
79 END
```

Routine Listing:

```
01 LBL "SR"
02 SIGN
03 SF 10
04 RDN
05 RCL b
06 STO M
07 RDN
08 FC?C 10
09 RTN
10 "├***"
11 RCL IND L
12 ISG L
13 ""
14 X<> IND L
15 STO N
16 "├**"
17 X<> IND L
18 STO O
19 "├**"
20 X<> O
21 STO a
22 X<> N
23 CLA
24 STO b
25 LBL "LR"
26 SIGN
27 RDN
28 "+"
29 RCL a
30 STO N
31 RDN
32 RCL b
33 X<> M
34 STO O
35 ASTO IND L
36 ISG L
37 ""
38 "├*****"
39 STO O
40 ASTO IND L
41 RDN
42 CLA
43 END
```

(95 bytes)

CHAPTER III

INITIALIZATION & PROMPTING

3-1 SIZE & PROGRAM TITLE SUBROUTINES ("TITLE", "SIZE?" & "T+S"):

Program Title Subroutine: A nice touch in many applications is a title on the printed output. This subroutine prints the title, double wide, and spaces appropriately; key in the title, then execute the routine:

LBL "TITLE", ADV, SF 12, FS? 55, PRA, CF 12, ADV, RTN. (20 bytes)

SIZE Check Subroutine: It can be very annoying to be on the last input of a long input sequence and get a "NONEXISTENT" error. This is usually the result of an incorrect SIZE. By executing this subroutine at the beginning of a program, this problem is eliminated:

LBL "SIZE?" , "SIZE>=", ARCL X, 1, -, SF 25, RCL IND X, RTN. (25 bytes)

Flag 25 is the Error Ignore Flag. To call this routine, you must place the necessary SIZE in X prior to the call. The calling sequence must not be in a subroutine. Follow the call with 'FC?C 25, PROMPT'. Example: if a minimum SIZE of 054 is required by a program, the sequence of steps in the initialization used to call "SIZE?" is: '54, XEQ "SIZE?", FC?C 25, PROMPT'.

Title and SIZE Combined: Since both routines may be needed, they can be combined:

```
01 LBL "T+S"    04 FC? 55    07 ADV          10 1        13 RCL IND X
02 ADV          05 PRA       08 "SIZE>="     11 -        14 RTN
03 SF 12        06 CF 12     09 ARCL X       12 SF 25                (33 bytes)
```

Example: The calling sequence for a Title of "F=MA" and a SIZE of 6 would be: '6, "F=MA", XEQ "T+S", FC?C 25, PROMPT'.

Source: Corvallis Division Column, PPC J, V6N7P19.

3-2 RESIZE? ("RS"): This routine tests to see if the current SIZE is great enough; if not, it prompts for the minimum SIZE needed. Have the minimum SIZE needed by the program in X before execution.

```
01 LBL "RS"     04 STO X          07 FS?C 25          10 ARCL Y
02 ENTER        05 SF 25          08 RTN              11 PROMPT
03 DSE X        06 VIEW IND X     09 "RESIZE: "       12 END      (33 bytes)
```

3-3 TEST SIZE: Tests if SIZE is great enough; have number of data registers needed in X before execution. Source: John Dearing (2791).

```
01 LBL "?S"       03 ARCL X    05 STO X    07 RCL IND X    09 PROMPT
02 "RESIZE: "     04 DSE X     06 SF 25    08 FC?C 25      10 END    (31 bytes)
```

3-4 SYNTHETIC TEST SIZE ("?S"): Displays the SIZE to be set in FIX 0, CF 29 (no decimal) mode, but restores the original display mode. Source: John Dearing (2791).

```
01 LBL "?S"       04 FIX 0      07 STO d    10 SF 25         13 PROMPT
02 "RESIZE: "     05 CF 29      08 DSE Y    11 RCL IND Y     14 END
03 RCL d          06 ARCL Y     09 ""       12 FC?C 25                (38 bytes)
```

3-5 SYNTHETIC VERIFY SIZE: To find if the current SIZE is great enough, key in the required SIZE, XEQ "VS"; routine prompts for a resize only if it is necessary. (If prompted, reSIZE as directed, then R/S). Contents of X, Y & Z registers (before required size number was keyed in) are returned; T is lost. To change to a version that returns execution to the main program, whatever the SIZE, without prompting, change step 09 (FS?C 25) to "FS? 25", and delete step 20 (PROMPT). Then execution of the routine in a program would be of this form: '..., XEQ "VS", FC?C 25, PROMPT, ...'. Source: Roger Hill (4940) (PPC ROM).

01 LBL "VS"	05 DSE T	09 FS?C 25	13 R↑	17 ARCL L	21 END
02 SF 25	06 ""	10 RTN	14 RCL d	18 STO d	
03 INT	07 RCL IND T	11 "RESIZE = "	15 FIX 0	19 RDN	
04 RDN	08 RDN	12 TONE 3	16 CF 29	20 PROMPT	(47 bytes)

3-6 INPUT ROUTINE ("IN"): This routine is used to prompt for, store, format and print input values. It uses R00 as a storage pointer. "IN" expects a 5-character (or less) input variable name in the Alpha Register when it is called [because appending an equal sign in step 06 creates an Alpha string of 6 characters, which is as long an Alpha string as the Y Register can hold (step 07)]. The format for calling this routine is shown by this example, which stores values in R06, R07 & R08: '..., 5, STO 00, "LEN.", XEQ "IN", "HT.", XEQ "IN", "WIDTH", XEQ "IN", ...'. Note that the number keyed in ('5' in this example) is one less than the number of the first register that will have a value stored in it (R06 in this example). Use '0' to start the loading in R01. "IN" is convenient for the user of your programs. Once a problem has been run, the user can rework the problem, keying only the values he or she wishes to change. (Pressing R/S without keying in a value when prompted leaves the value unchanged). This allows rapid sensitivity analysis of chosen variables. Flag 22 is set upon return from "IN" if the user made an input; it is clear if the user did not make an input. You may be able to make use of this fact. Note that this version of "IN" doesn't work if the printer is plugged in but is turned OFF. Source: Corvallis Division Column, PPC J, V6N7P18.

01 LBL "IN"	05 RCL IND 00	09 CF 21	13 ARCL Y	17 FC? 55	21 END
02 CF 22	06 "⊢="	10 AVIEW	14 STOP	18 RTN	
03 1	07 ASTO Y	11 SF 21	15 STO IND 00	19 ARCL X	
04 ST+ 00	08 "⊢?"	12 CLA	16 FS? 22	20 PRA	(44 bytes)

The version below replaces steps 07-14 above with "PROMPT". The question mark won't appear in the prompt, but input variable names need not be limited to 5 or fewer characters. Also, it can be used when the printer is plugged in but is OFF, if you CF 21 first.

01 LBL "IN"	04 ST+ 00	07 PROMPT	10 FC? 55	13 PRA
02 CF 22	05 RCL IND 00	08 STO IND 00	11 RTN	14 END
03 1	06 "⊢="	09 FS? 22	12 ARCL X	(31 bytes)

This last version will print old values (retained by skipping the prompt with R/S), as well as new values. If no printer is plugged in, it will pause to display the labeled value (new or old), then prompt for the next value. If the printer is plugged in but is OFF, CF 21 before execution and the routine will behave as in the version above. Source: John Dearing (2791).

01 LBL "IN"	04 ST+ 00	07 PROMPT	10 FS? 55	13 RTN	16 END
02 CF 22	05 RCL IND 00	08 STO IND 00	11 PRA	14 AVIEW	
03 1	06 "⊢="	09 ARCL X	12 FS? 55	15 PSE	(33 bytes)

3-7 OUTPUT ROUTINE ("OUT"): This routine formats and either prints or displays the value in the X Register. Put the values to be output in X and the name of the value in the Alpha Register before executing "OUT". Routine sets Flag 21. Here is an example of the use of "OUT": '..., RCL 06, "LENGTH", XEQ "OUT", RCL 07, "HEIGHT", XEQ "OUT", RCL 08, "WIDTH", XEQ "OUT", ...'. Source: Corvallis Division Column, PPC J, V6N7P18.

LBL "OUT", SF 21, "⊢=", ARCL X, AVIEW, RTN. (16 bytes)

3-8 YES OR NO QUESTION SUBROUTINE ("YN"): It is frequently desirable to ask the user a question with two possible answers. It is almost always possible to pose the question in a 'yes' or 'no' context. It is usually desirable to remember the user's answer in the form of a set (yes) or clear (no) flag. The routine "YN" aids in asking such questions. (1) It adds the characters "? Y/N" to the prompt put in the Alpha Register prior to call. Note that the prompt must contain six or fewer characters. (2) The routine prints the results of the question if a printer is plugged in and is ON. (3) If a printer is not plugged in, the routine pauses to display the results of the question; if a printer is plugged in but is OFF, CF 21 first. (4) The routine sets or clears the flag specified by the contents of the X Register on call (if X=5, Flag 05 is set or cleared). (5) The routine retains the current status of the flag if the user fails to answer the question. (6) The routine sets and clears Alpha Mode as needed. Example: a program might ask a user if units to be used are metric (SI) or English: '..., 0, "METRIC", XEQ "YN", ...'; if the units to be used are metric, the user keys "Y", and the routine sets Flag 00; if he presses "N", Flag 00 will be cleared. This flag can be tested later in the program as needed. NOTE: This routine could be modified to accept answers other than Yes or No; for example, Left/Right (tails of a normal curve), or Upper/Lower, or even a pair of numbers (1 or 2). Source: Corvallis Division Column, PPC J, V6N7P19.

01 LBL "YN"	07 AOFF	13 "Y"	19 "⊢: "	25 AVIEW	31 END
02 CF 23	08 FC? 23	14 ASTO Y	20 FS? IND T	26 FC? 21	
03 ASTO L	09 RTN	15 X=Y?	21 "⊢YES"	27 PSE	
04 "⊢? Y/N"	10 CF IND X	16 SF IND T	22 FC? IND T	28 FC? 21	
05 AON	11 RDN	17 CLA	23 "⊢NO"	29 RTN	
06 PROMPT	12 ASTO X	18 ARCL L	24 FC? 21	30 PRA	(69 bytes)

3-9 SHORT YES/NO QUESTION: '..., "(question)?", CF 23, AON, PROMPT, AOFF, ..., F? 23 (any test), ...'. If the answer is 'yes', press "Y", R/S; if the answer is 'no', just press R/S (any Alpha characters will do in place of "Y"). The status of Flag 23 (the Alpha entry flag) records the answer; if 'yes', Flag 23 is set; if 'no', Flag 23 is clear (until another Alpha entry is made); test Flag 23 to decide what to do. The question could be "PRINT?" or "ANY CHANGES?" for example, and need not be six or fewer characters. Source: Valentin Albillo (4747).

3-10 A PROMPT AFTER INITIALIZATION: Terminate long initializations with "READY" and/or a tone or BEEP. A better prompt than "READY" might be one that tells the user what to do: for example, if you are to enter X, Y and f (frequency) in the stack, then press A, your prompt might be: "X,↑,Y,↑,F: A".

3-11 FLAG DETERMINES PROMPT: ..., CF 00, ..., "MSG1", FS? 00, "MSG2", PROMPT, ... If Flag 00 is clear, "MSG1" will be the prompt; if Flag 00 is set, "MSG2" will appear. An example (where Flag 00 is cleared in the initialization): LBL C, SF 00, LBL B, "SLOPE?", FS? 00, "ANGLE?", PROMPT, FS?C 00, TAN, ... If you press B, the prompt is "SLOPE?"; if you press C, the prompt is "ANGLE?"; Flag 00 is then cleared.

3-12 "PROMPT X": To avoid printing (printer plugged in and ON) and to avoid stopping (printer plugged in and OFF) when a stack or numeric register must be viewed, use the Alpha Register and PROMPT. For example, instead of VIEW X, use 'CLA, ARCL X, PROMPT'. The display mode can be reset before the program stops for the PROMPT by inserting, for example, a FIX 2 just before the PROMPT. Source: John Dearing (2791) (PPC CJ, V7N9P28). See 2-31, 4-1, 4-18, 6-4, 6-6, 6-14, 22-23.

3-13 HAS A NEW NUMBER BEEN KEYED IN? IF NOT, USE OLD ONE: ..., FS?C 22, STO 01, RCL 01, ... Flag 22 must be cleared before possible input.

3-14 TO DETECT NUMERIC INPUT: The usual method is ..., CF 22, "(question)?", PROMPT, F? 22 (any test), ... If a number is keyed in, Flag 22 is set; if you only pressed R/S, Flag 22 is clear. You then test Flag 22 to decide what to do. If your data cannot include 0 as an input, there is a better method that saves two bytes: ..., 0, "(question)?", PROMPT, X=0? (or other test), ... If you input some data, the test against zero gives a different result than if you just press R/S without an entry. Source: Valentin Albillo (4747).

3-15 REVIEW OLD ENTRY BEFORE KEYING NEW ONE: Insert 'RCL nn' prior to PROMPT when prompting for an input to be subsequently stored in that same register. Example: ..., "HEIGHT?", RCL 01, PROMPT, STO 01, ... This way the previous value stored in that register can be reviewed just by pressing backarrow (the correction key) after the prompt appears. If the old value is to be used again, just press R/S; if a new value is to be stored, key it, then press R/S. Source: Robert McDonald (5460).

3-16 INPUTTING IN ONE FORM, USING IN ANOTHER: Using an input routine (as "IN", 3-6) where the old value is to be used if a new value is not input, but where values input are in one form (say 'feet'), but are to be used in another form (say 'meters'): use register arithmetic to convert old value to units of input, then prompt with input routine, then use register arithmetic to convert back to units used by program. For example:

..., "L, FEET", .3048, ST/ 04, XEQ "IN", .3048, ST* 04, ...

This converts the contents of R04 to feet, then prompts for an input in feet; if you want to use the previous value input, just press R/S; otherwise, key new value, then R/S. After returning from "IN", contents of R04 (new or old value) is converted to meters.

3-17 CALLING DIFFERENT FUNCTIONS: For programs that need to call different functions at different times, you can have it ask for the name of the function; it will then store the name and execute it indirectly as needed. The function needed can be keyed in just before running the program, or it can have been programmed earlier. To key it in just before running the main program, press 'SHIFT GTO .. PRGM', then enter the function; next, switch out of PRGM Mode and execute the main program. Your function(s) must have a global label of six or fewer Alpha characters.

..., "FUNCT. NAME?", AON, PROMPT, AOFF, ASTO 03, ..., XEQ IND 03, ...

3-18 PROMPT FOR INPUT WITHOUT STOPPING PROGRAM ("NUM?" & "WRD?"): These two subroutines can be used for requesting input without stopping the running program. The calling program provides a prompting message in the Alpha Register, and the subroutine provides a steady display while waiting for a response. After the response, control is returned to the calling program. Load the prompting message into the Alpha Register; for numeric input, XEQ "NUM?"; for Alpha input, XEQ "WRD?". Control is returned to the calling program with the numeric response in the X Register or the Alpha response in the Alpha Register. Source: HP KEY NOTES, V4N1P6.

```
01 LBL "NUM?"   05 PSE                :  01 LBL "WRD?"   05 LBL 02     09 AOFF
02 AVIEW        06 FC?C 22            :  02 CF 23        06 PSE        10 RTN
03 CF 22        07 GTO 01             :  03 AON          07 FC?C 23
04 LBL 01       08 RTN    (20 bytes)  :  04 AVIEW        08 GTO 02     (21 bytes)
```

CHAPTER IV

DISPLAY

4-1 VIEW ALPHA ("VA"): This routine, unlike AVIEW, never causes the program to stop. "VA" may be followed with a STOP or PAUSE (which may be controled by flags). "VA" prints Alpha if the printer is ON and Flag 21 is set.

01 LBL "VA"	03 PRA	05 FS?C 21	07 AVIEW	09 SF 21	
02 SF 25	04 SF 25	06 CF 25	08 FC?C 25	10 RTN	(24 bytes)

Source: Roger Hill (4940) (PPC ROM). See 2-31, 3-12, 4-18, 6-4, 6-6, 6-14, 22-23.

4-2 GOOSE VS. (A)VIEW: The flying goose character will disappear from the display after a program executes VIEW or AVIEW, and the contents of the register being viewed will remain in the display until the next CLD, VIEW, AVIEW or STOP is executed. CLD and STOP will return the goose to the display. Source: 'HP-41C Operating Manual', © Copyright (June 1980) Hewlett-Packard Company. Reproduced with permission.

4-3 DISPLAY MODE SAVE AND RECALL ("DSPS" & "DSPR"): "DSPS" will save the number of digits displayed and the display mode. The number in the T Register is replaced by '8'. The contents of the X, Y & Z Registers are unchanged. "DSPR" will recall this previously saved display setting without affecting the stack. These routines use Flags 05 & 06 and Register 00. "DSPS" & "DSPR" are useful when a subroutine displays a number in a certain format, and you want to return to the format used when the subroutine was called. Source: Scott Morrison (4360) (PPC J, V6N5P31).

01 LBL "DSPS"	08 RDN	15 ST+ 00	22 FS? 40	28 LBL "DSPR"	34 FS?C 06
02 0	09 2	16 RDN	23 SF 05	29 FS? 05	35 FS? 05
03 STO 00	10 FS? 38	17 8	24 FS? 41	30 FIX IND 00	36 SCI IND 00
04 RDN	11 ST+ 00	18 FS? 36	25 SF 06	31 FS? 06	37 END
05 1	12 RDN	19 ST+ 00	26 RDN	32 ENG IND 00	
06 FS? 39	13 4	20 CF 05	27 RTN	33 FC?C 05	(75 bytes)
07 ST+ 00	14 FS? 37	21 CF 06			

4-4 SYNTHETIC STORE & RECALL DISPLAY MODE ("SD" & "RD"): "SD" stores the current display mode in the register pointed to by the integer in X; "RD" recalls the display mode previously stored in the register pointed to by X. Both routines restore the values that were in the X, Y & Z Registers before the pointer was keyed in. Source: Keith Jarett (4360) (PPC ROM). "SD" saves Flags 16-55.

01 LBL "SD"	06 "├◆◆"	11 ASTO IND L	16 ARCL IND L	21 X<> O	26 RDN
02 SIGN	07 X<> M	12 RDN	17 RDN	22 STO N	27 CLA
03 RDN	08 "*"	13 RTN	18 RCL d	23 "├*****"	28 END
04 RCL d	09 X<> M	14 LBL "RD"	19 STO N	24 X<> N	
05 STO M	10 STO N	15 SIGN	20 "├**"	25 STO d	(66 bytes)

4-5 SYNTHETIC SAVE DISPLAY TEMPORARILY: Use Status Register d, the flag register: recall it just before a temporary display change, store it back as soon as possible. For example, to put the contents of the X Register in 'FIX 0, CF 29' mode into the Alpha Register, then recover the previous display mode, use the following: ..., RCL d, FIX 0, CF 29, ARCL Y, STO d, ... Leaves 'garbage' in X.

4-6 LENGTH OF ALPHA STRINGS: Analyze Alpha strings, including PROMPTs, using graph paper, keeping in mind these lengths:

6: As many characters as can be stored in a numeric or stack register.

9: As many as will show in a 2-digit line number (01-99) when in PRGM mode without scrolling; as many as can be appended to a full 15-character string without losing characters on the left.

12: As many characters as will show with a PROMPT without scrolling (not counting nonadjacent periods, commas or colons).

15: As many characters as will fit in one line of a program.

24: As many characters as will fit into the Alpha Register.

4-7 DISPLAY ONE TEXT OR ANOTHER DEPENDING ON A TEST OR FLAG: "TEXT1", TEST, "TEXT2". For example: "RIGHT", X=Y?, "WRONG" places "WRONG" in the Alpha Register if X=Y, but "RIGHT" if X≠Y. This technique may be used similarly to display long messages more economically: for example, 'CLA, X≠Y?, "IN", "├CORRECT"' places "CORRECT" in the Alpha Register if X=Y, and "INCORRECT" if X≠Y. Source: Valentin Albillo (4747).

4-8 SYNTHETIC DISPLAY SET ("DS"): This routine gives the HP-41C/V a "DSP" function; when "DS" is executed, the calculator stays in the "FIX", "SCI" or "ENG" half of the display mode, but "DS" uses the absolute value of the integer portion of the number in X (if the result is between 0 and 9, inclusive) to determine the number of significant digits after the first one to be displayed. "DATA ERROR" results with an input outside of this range. Example: in FIX 2 Mode, key '6', XEQ "DS"; calculator is now in FIX 6 Mode. Routine destroys the contents of the T & L Registers. Source: Keith Jarett (4360) (PPC ROM).

01 LBL "DS"	04 RCL d	07 X<> d	10 X<> O	13 X<> N	16 CLA
02 SIGN	05 STO O	08 STO M	11 STO M	14 STO d	17 END
03 RDN	06 SCI IND L	09 "├*****"	12 "├**"	15 RDN	(42 bytes)

4-9 SYNTHETIC CHANGE FIX-, SCI-, OR ENG- HALF OF DISPLAY MODE ("7FIX", "7SCI" & "7ENG"): This routine simulates the HP-67/97 versions of "FIX", "SCI" & "ENG": the number of displayed digits doesn't change. To use, XEQ "7FIX" to change to a FIX mode with the same number of digits displayed as before; XEQ "7SCI" or "7ENG" to change to a SCI or ENG mode with the number of digits displayed unchanged. This routine uses no numeric data registers and doesn't disturb the stack (including L). It does not change the status of any flag other than Flags 40 & 41 (which select FIX, SCI or ENG). The Alpha Register is used, then cleared. Executes in 1 second. Source: Valentin Albillo (4747).

01 LBL "7FIX"	07 STO N	13 STO d	19 GTO 01	25 STO M	31 END
02 XEQ 02	08 CLX	14 CF 00	20 LBL "7ENG"	26 "├**"	
03 SF 00	09 RCL d	15 CF 01	21 XEQ 02	27 X<> N	
04 GTO 01	10 STO M	16 RTN	22 SF 01	28 STO d	
05 LBL 02	11 "├*****"	17 LBL "7SCI"	23 LBL 01	29 X<> O	
06 CLA	12 X<> M	18 XEQ 02	24 X<> d	30 CLA	(85 bytes)

4-10 SYNTHETIC 'FIX/ENG' DISPLAY MODE: Setting Flags 40 and 41 simultaneously puts the 41C/V in 'FIX/ENG' display mode. In ordinary 'FIX' format (Flag 40 set, Flag 41 clear), numbers which are too large or too small to display properly cause the display to default to the 'SCI' format; in 'FIX/ENG' format, however, the default is to the 'ENG' mode. Source: William Wickes (3735) ('Synthetic Programming on the HP-41C').

4-11 APPROXIMATING CONTINUOUS SCROLLING: Continuous scrolling to the left can be approximated by overlapping register recalls. Example: ..., ARCL 01, ARCL 02, ARCL 03, ARCL 04, AVIEW; ARCL 03, ARCL 04, ARCL 05, ARCL 06, AVIEW; etc. Source: Richard J. Nelson (1) (PPC J, V6N5P32).

4-12 SCROLLING READABILITY: Leave a blank space or two at the beginning of 13 or 14 character displays for better readability of the scrolled message. Source: Richard J. Nelson (1) (PPC J, V6N5P32).

4-13 SCROLL LEFT ("SCE"): Especially for strings greater than 24 characters in length. (1) Write out the message in full, then mark it off into groups of 13 characters (nonadjacent periods, commas and colons don't count; spaces do). Use leading blanks if you wish. (2) In PRGM Mode, key a label if appropriate, then turn Alpha Mode ON. (3) Enter the first 13 characters. (4) Rapidly press 'ALPHA, ALPHA, ALPHA' to terminate the program line and prepare for the next line. (5) Press 'SHIFT APPEND' and then enter the next 11 characters (13 + 11 = 24). (6) Press "AVIEW". (7) Beginning with the first character used in step 5, go to step 3. (8) Repeat steps 3-7 until the message is complete. Example: the following routine displays "**HP-41C/V **" repeatedly scrolling across the display from right to left. Step 02 is 12 blanks followed by one "*"; step 05 is "*HP-41C/V**" followed by 2 blanks; step 06 is append 11 blanks. Source: David Walker (1840) (PPC J, V6N7P3).

01 LBL "SCE"	03 "⊢*HP-41C/V**"	05 "*HP-41C/V** "	07 AVIEW
02 " *"	04 AVIEW	06 "⊢ "	08 GTO "SCL"

4-14 SCROLL RIGHT (GOOSE REPLACEMENT) ("SCR" & "SO"): This routine takes advantage of a minor, good bug in the 41C/V involving the error flag. Put the desired replacement character(s) in the Alpha Register, set Flag 25, AVIEW, and set any non-existent flag. For example, put these steps into a program (say before a loop, where Alpha won't be disturbed) to replace the goose with a hyphen: "-", SF 25, AVIEW, SF 99. Here's a demonstration routine which will prompt you for the Alpha string to use (try "GOOSE"):

01 LBL "SCR"	04 STOP	07 AVIEW	10 1.2	13 END
02 "SCROLL CHAR.?"	05 AOFF	08 SF 99	11 SIN	
03 AON	06 SF 25	09 LBL 01	12 GTO 01	(39 bytes)

It appears that the Alpha Register is scrolled. Now try "AAAAAAAAAAAATESTTESTTEST" (12 'A's + 3 'TEST's). You will hear a tone when the 24th character is keyed in. The display first scrolls to the left, then to the right. The A's disappear and only "TESTTESTTEST" scrolls. Also observe that the scroll wraps around and all 12 display characters are always in the display. Conclusion: this routine scrolls the last 12 characters after a normal 'read scroll' to the left, if the Alpha string is more than 12 characters. For fun, try six pairs of any of the following for visual effect: "XY", "MW", "+-", & ":.".

You can use a subroutine to replace the goose; at the beginning of a series of loop calculations, place the desired display string in the Alpha Register, then XEQ "SO". Have the following routine available: LBL "SO", SF 25, AVIEW, SF 99, RTN. Source: Richard Nelson (1) (PPC J, V6N8P24).

4-15 SYNTHETIC GOOSE REPLACEMENT: With an Alpha string of up to 12 characters in Alpha, put the instruction sequence 'RCL d, AVIEW, STO d' into a program (say before a loop), and the goose will be replaced by the contents of the Alpha Register, stepping around the display. Source: William Wickes (3735) ('Synthetic Programming on the HP-41C').

4-16 DISPLAY X & Y SIMULTANEOUSLY ("XY" & "X?Y"): This routine is useful when two numbers are output (complex numbers or coordinates, for example). "XY" uses the current display mode; "X?Y" formats according to 'm.n' stored in R00, setting X to FIX m and Y to FIX n. Source: Peter Ladrach (5060) (PPC CJ, V7N4P6).

```
01 LBL "XY"    05 ARCL Y    :  01 LBL "X?Y"    05 "⊦ "     09 *           13 AVIEW
02 CLA         06 AVIEW     :  02 CLA          06 RCL 00   10 FIX IND X   14 RTN
03 ARCL X      07 RTN       :  03 FIX IND 00   07 FRC      11 RDN
04 "⊦ "        (16 bytes)   :  04 ARCL X       08 10       12 ARCL Y        (27 bytes)
```

4-17 SYNTHETIC DISPLAY TEST ("DT"): To test all display annunciators, XEQ "DT"; the routine will pause to display 12 commas, then stop to display 12 boxed stars and 12 colons, plus all the lower annunciators (BAT, USER, etc). To clear the display and restore the display mode, press PRGM to get out of PRGM Mode, then R/S. The routine uses the T & L Registers. Lines 02 and 10 are nonstandard; Line 02 is decimal 247, 248, 0, 0, 16, 0, 33, 232; Line 10 is decimal 246, 128, 58, 128, 58, 128, 58. Source: William Wickes (3735) & Valentin Albillo (4747) (PPC ROM).

```
 01 LBL "DT"             05 ASTO L    *10 ":::"     15 X<> d     20 CLD
*          02 "◆◆θ◆!     06 ARCL L     11 ASTO L    16 AVIEW     21 END
"                        07 AON        12 ARCL L    17 STOP
 03 RCL M                08 PSE        13 ARCL L    18 X<> d
 04 ",,,,,,"             09 AOFF       14 ARCL L    19 RDN         (56 bytes)
```

4-18 "AVIEW" REPLACEMENT ROUTINES ("AV" & "AVN"): These routines can be used in place of "AVIEW" in a program. "AV" (Alpha View, stop only if printer is off) has the following characteristics: a. No printer--simply AVIEW without stopping. b. Printer is off, and "PRINTER OFF" is displayed--merely turn on the printer and press R/S to print and display the Alpha Register. c. Printer is on--it prints and displays the Alpha Register without stopping. d. Flag 21--Flag 21 does not control the printer and retains its set or clear status. "AV" was written to aid users who normally operate their HP-41 system with the philosophy that if their printer is connected it should print, and they should be reminded to turn it on if it is off. "AV" illustrates another use of flags. Lines 02 and 03 use Flag 14 (the flag that allows you to record on a clipped corner card) to store the status of Flag 21. Lines 11 and 12 restore both flags to their original status. There is little danger in using Flag 14 in this way because it is very unlikely that you will stop the routine to record on a clipped-corner card.

"AVN" (Alpha View, never stop): this routine never causes a STOP. It is similar to "VA" (4-1), but this routine will print if the printer is on, even if Flag 21 is clear, while "VA" won't print if Flag 21 is clear.

Source: HP KEY NOTES, V5N1P7. See 2-31, 3-12, 4-1, 6-4, 6-6, 6-14, 22-23.

```
01 LBL "AV"     08 LBL 14                 01 LBL "AVN"    08 CF 21
02 FS? 21       09 CF 21                  02 FS? 21       09 AVIEW
03 SF 14        10 AVIEW                  03 SF 14        10 FS?C 14
04 FC? 55       11 FS?C 14                04 SF 21        11 SF 21
05 GTO 14       12 SF 21                  05 SF 25        12 END
06 SF 21        13 END                    06 PRA
07 PRA              (29 bytes)            07 CF 25            (29 bytes)
```

CHAPTER V

ALPHA MANIPULATIONS

5-1 ALPHA TO MEMORY & MEMORY TO ALPHA ("AM" & "MA"): With a control number bbb.eee in X, XEQ "AM" to store the contents of the Alpha Register in data registers, or XEQ "MA" to recall data registers into the Alpha Register. "AM" clears the Alpha Register--restore with the same control number and "MA". The end register (eee) should be no more than 3 higher than the beginning register (bbb) (unless using the full form of the control number, bbb.eeeii). For example, key '17.020', XEQ "AM" to store all 24 characters of the Alpha Register in R17 - R20; using '17.018' as the control number instead will store the first 12 characters of the Alpha Register only. Remember to have a control number in X before executing either routine. Source: Keith Jarett (4360) (PPC ROM).

01 LBL "AM"	04 ASHF	07 RTN	10 LBL 02	13 GTO 02
02 LBL 01	05 ISG X	08 LBL "MA"	11 ARCL IND X	14 END
03 ASTO IND X	06 GTO 01	09 CLA	12 ISG X	(32 bytes)

5-2 ALPHA STRING TESTING RESTRICTIONS ON EARLY MACHINES: If you are testing two Alpha strings that were originally longer than six characters (when created in the Alpha Register), then you must perform the following procedure to ensure proper string truncation and test results. Strings can only be tested with X=Y? or X≠Y?. (1) Store the first string into a register using ASTO nn. If the string is not longer than six characters, skip this step and go to step 04. (2) Clear the Alpha Register with CLA. (3) Recall the string into the Alpha Register using ARCL nn. (4) Store the string into the X Register using ASTO X. (5) Repeat the steps above for the second Alpha string, but store it in the Y Register using ASTO Y. (6) Execute "X=Y?" or "X≠Y?". Source: HP KEY NOTES, V4N1P3.

5-3 CAUTION WHEN EDITING ALPHA BLANKS IN COMBINATION WITH PUNCTUATION (. : ,): Apparently the backarrow (correction) key causes the underscore prompting mark to move back two characters in the display while only one character is actually removed from the Alpha Register. Try these steps in ALPHA Mode: 'CLA, "Z", SPACE, COMMA, BACKARROW, BACKARROW, AVIEW'. You will see the "Z" disappear from the display but an Alpha Register call (or ALPHA, ALPHA) shows that it is still in the Alpha Register. For a more spectacular display of this effect, see what happens with more and more erasures of 'SPACE, COMMA' pairs following the "Z". Eventually, repeated backarrow erasures cause the underscore mark to disappear from the left of the display, reappear on the right of the display two strokes later, then apparently recover the remaining Alpha contents, and continue as before. Source: Charles Harris (1959) (PPC CJ, V7N3P28).

5-4 SYNTHETIC DELETE LAST ALPHA CHARACTER ("AD"): This routine deletes the last character of the Alpha string in the Alpha Register. Uses the stack. Source: Gerard Westen (4780) (PPC ROM).

01 LBL "AD"	05 X<> N	09 STO M	13 RDN	17 .1	21 END
02 RCL P	06 "⊢******"	10 ASTO X	14 STO N	18 STO P	
03 RCL O	07 RCL N	11 RDN	15 RDN	19 ASHF	
04 .	08 CLA	12 STO M	16 STO O	20 ARCL Z	(47 bytes)

5-5 SYNTHETIC ISOLATE & SUBSTITUTE CHARACTERS ("NC" & "SU"): "NC" (Nth Character) isolates the nth character from the right of the string in Alpha. It assumes a positive number in X whose integer portion, n, is from 1 to 10. It replaces an arbitrarily long string in Alpha with its nth character from the right; it also places that character into X. The values in X & Y before keying in the number are returned to Y & Z. "SU" (Substitute Character) provides a string-editing capability; with a positive number in X whose integer portion, n, is from 1 to 10, this routine replaces the nth character from the right in Alpha with the character in Y (with the rightmost character in Y, if more than one). Values in X & Y before keying in the number are returned to X & Y. An integer 1 greater than the number of characters in Alpha adds the character in Y onto the left of the Alpha string. Source: William Wickes (3735) (PPC ROM).

01 LBL "NC"	11 RCL d	21 GTO 14	31 DSE L	41 CLX	51 LBL 14
02 CF 25	12 SCI IND Y	22 X<> Z	32 CLX	42 ISG L	52 X<> O
03 GTO 14	13 ARCL Y	23 STO O	33 X<> L	43 CLX	53 CLA
04 LBL "SU"	14 STO d	24 "├******"	34 10↑X	44 X<> P	54 STO M
05 SF 25	15 RDN	25 X<> Z	35 RCL d	45 STO N	55 ASTO X
06 LBL 14	16 X<> O	26 STO P	36 FIX 0	46 CLX	56 END
07 INT	17 FS? 25	27 RDN	37 CF 29	47 X<> O	
08 E1	18 RCL P	28 X<> O	38 ARCL Y	48 STO M	
09 X<>Y	19 "├*"	29 X<> N	39 STO d	49 RDN	
10 -	20 FC?C 25	30 STO M	40 RDN	50 RTN	(112 bytes)

5-6 SYNTHETIC CHARACTER-DECIMAL CONVERSIONS ("CD" & "DC"): "CD" (Character to Decimal): With a single Alpha character in the Alpha Register, this routine will return the corresponding decimal number to X (0-255), according to the Byte Table. With more than one Alpha character in Alpha, the decimal equivalent of the rightmost character is returned. With Flag 10 clear, the Alpha character will be deleted from the Alpha Register; with Flag 10 set, it will be left in Alpha. Values in X, Y & Z Registers before execution will be returned to Y, Z & T. "DC" (Decimal to Character): With a positive number in X whose integer portion is 0-255, this routine will add the corresponding Alpha character to the Alpha string in the Alpha Register. The values in X & Y before keying the decimal will be returned to X & Y; Z & T values are lost. Source: William Wickes (3735) & Roger Hill (4940) (PPC ROM).

01 LBL "CD"	12 "├***"	23 ST* L	34 +	45 FS? 06	56 STO P
02 "├◆↓****"	13 X<> M	24 X<> L	35 OCT	46 SF 08	57 RDN
03 RCL M	14 X<> L	25 ST+ O	36 X<> d	47 X<> d	58 X<> O
04 FS? 10	15 X<> N	26 CLX	37 FS?C 11	48 X<> M	59 X<> N
05 "├*"	16 INT	27 X<> O	38 SF 12	49 RCL N	60 STO M
06 STO M	17 ST+ O	28 RTN	39 FS?C 10	50 "├*"	61 RDN
07 CLX	18 RDN	29 LBL "DC"	40 SF 11	51 X<> O	62 END
08 X<> O	19 6	30 INT	41 FS?C 09	52 X<>Y	
09 SIGN	20 ST* O	31 256	42 SF 10	53 STO N	
10 CLX	21 RDN	32 MOD	43 FS? 07	54 X<> P	
11 X<> N	22 E1	33 LASTX	44 SF 09	55 "├*"	(129 bytes)

5-7 SYNTHETIC HEX-NNN CONVERSIONS ("NH" & "HN"): An 'NNN' is a nonnormalized number--one whose sign nybble is other than 0 (a positive number), 1 (an Alpha string), or 9 (a negative number), or one whose sign nybble is 0 or 9, but which contains any digits of value A - F. "HN" (Hex to NNN): Changes a hex number in Alpha (up to 7 hex digits long) to a NNN in X, and to its corresponding Alpha string in Alpha. Values in X, Y & Z before execution are returned to Y, Z & T; zero is returned to L. Attempting an arithmetic operation on an NNN gives "ALPHA DATA" error message. "NH" (NNN to Hex): Converts an NNN in X to Hex in Alpha (and sets ALPHA Mode). The NNN remains in X. "NH" will also convert an Alpha string of up to 6 characters in X to hex in Alpha. The Alpha string remains in X. Source: Roger Hill (4940), William Wickes (3735) & John McGechie (3324).

```
01 LBL "NH"
02 CLA
03 STO M
04 SIGN
05 X<> d
06 "├◆◆*"
07 .
08 X<> M
09 "├◆◆"
10 X<> M
11 FIX 9
12 ARCL N
13 "├**"
14 ARCL O
15 X<> O
16 FIX 3
17 ARCL O
18 STO O
19 "├*"
20 RDN
21 STO d
22 CLX
23 FS? 10
24 GTO 12
25 RDN
26 14
27 LBL 01
28 RCL O
29 X<> d
30 FC? 06
31 FS? 05
32 FC? 04
33 GTO 14
34 LBL 13
35 SF 01
36 CF 02
37 CF 03
38 CF 04
39 FS?C 07
40 GTO 14
41 SF 07
42 FS?C 06
43 GTO 14
44 SF 06
45 CF 05
46 LBL 14
47 X<> d
48 STO M
49 "├*"
50 RDN
51 DSE X
52 GTO 01
53 LBL 12
54 STO P
55 X<> O
56 X<> N
57 STO M
58 X<> L
59 AON
60 RTN
61 LBL HN"
62 7
63 SIGN
64 LBL 02
65 RDN
66 RCL N
67 X<> d
68 CF 11
69 CF 10
70 FC?C 09
71 GTO 14
72 SF 12
73 FC?C 15
74 SF 15
75 FS? 15
76 GTO 14
77 FC?C 14
78 SF 14
79 FC? 14
80 SF 13
81 LBL 14
82 FS? 07
83 SF 11
84 FS? 06
85 SF 10
86 FS? 05
87 SF 09
88 FS? 04
89 SF 08
90 FC? 01
91 GTO 14
92 SF 08
93 FC?C 11
94 SF 11
95 FS? 11
96 GTO 14
97 FC?C 10
98 SF 10
99 FC? 10
100 SF 09
101 LBL 14
102 X<> d
103 X<> O
104 "├*"
105 STO P
106 "├*"
107 X<> P
108 STO O
109 DSE L
110 GTO 02
111 CLA
112 STO M
113 AOFF
114 END
```

(227 bytes)

5-8 TO CONVERT A NUMBER IN X (6 OR FEWER DIGITS) INTO ITS ALPHA FORM IN X ("N-A"): LBL "N-A", ARCL X, ASTO X, CLA, RTN. Source: Jake Schwartz (1820) (PPC J, V6 N8P26).

5-9 LOSING CHARACTERS WITH ARCL: If "ARCL" adds characters to a full Alpha Register, the leftmost characters are lost, with no tone warning. Source: William Wickes (3735).

CHAPTER VI

FLAGS & TONES

6-1 FLAG TOGGLING ("FT"): The instruction sequence 'FC?C nn, SF nn' will set a flag if clear or clear a flag if set (toggle the flag). It may be used in a loop. The following routine, "FT", will toggle the flag whose number (0-29) is in X: LBL "FT", FC?C IND X, SF IND X, RTN. Source: Ron Knapp (618) (PPC J, V6N5P6) & Jake Schwartz (1820) (PPC J, V6N8P26).

6-2 SET OR CLEAR A FLAG WITH 0 OR 1: LBL A, SF 01, X=0?, CF 01, ... Source: Bill Kolb (265) (BP 67/97).

6-3 CLEAR MULTIPLE FLAGS ("CFX" & "CFA"): "CFX" clears a 'block' of flags, as determined by the (bbb.eee) control number in X before execution. For example, to clear Flags 5-10, key '5.01', XEQ "CFX". "CFA" supplies 0.025 to "CFX" to clear Flags 0-25. These routines use no numeric data registers and preserve X, Y, Z & L Registers. Source: William Cheeseman (4381) (PPC CJ, V7N5P7).

LBL "CFA", .025, LBL "CFX", CF IND X, ISG X, GTO "CFX", RDN, RTN. (29 bytes)

6-4 SYNTHETIC INVERT FLAG ("IF"): This routine changes the setting of the flag whose number is in X. It works for flags 00-29, 31-44, 47-50, 52 & 55. Flag 47 controls SHIFT Mode, Flag 48 controls ALPHA Mode, Flag 50 stops and views the goose (press backarrow [correction] key to clear), Flag 52 controls PRGM Mode. Key '52', XEQ "IF" and PRGM Mode will be set at the last line of the routine (if the routine ends with RTN, RTN or with RTN, END). The values in X and Y (before keying flag number) are restored; Z & T are lost. Note on Flag 55: "IF" will toggle Flag 55 if the printer is NOT plugged in, but it won't work if the printer IS plugged in (Flag 55 always tests set if the printer is plugged in). Source: Roger Hill (4940) (PPC ROM).

```
01 LBL "IF"
02 ABS
03 24
04 +
05 STO M
06 8
07 ST/ M
08 MOD
09 RCL d
10 X<> M
11 INT
12 SCI IND X
13 ARCL X
14 X<>Y
15 X<> O
16 X<> N
17 X<> d
18 FC?C IND O
19 SF IND O
20 X<> d
21 STO M
22 RDN
23 12
24 -
25 SCI IND X
26 ARCL X
27 X<> O
28 STO d
29 RDN
30 CLA
31 RTN
32 END
```
(58 bytes)

6-5 SYNTHETIC VIEW FLAGS ("VF"): XEQ "VF" to see which flags are set. "IF", executed in line 04, is routine 6-4 above. Line 08 is nonstandard; it is decimal 245, 4, 168, 0, 128, 1. Source: Keith Jarett (4360) & Roger Hill (4940) (PPC ROM).

```
 01 LBL "VF"
 02 50
 03 FC? 50
 04 XEQ "IF"
 05 "βP◆◆◆◆"
 06 RCL M
 07 SIGN
*08 "α    ◆×"
 09 X<> M
 10 4
 11 RDN
 12 "FLAGS SET:"
 13 X<> d
 14 XEQ "VA"
 15 X<> d
 16 CLA
 17 LBL 01
 18 FS? IND L
 19 XEQ 02
 20 ISG L
 21 GTO 01
 22 X<> d
 23 FC?C 24
 24 XEQ "VA"
 25 ADV
 26 BEEP
 27 X<> d
 28 RDN
 29 RTN
 30 LBL 02
 31 "⊢ "
 32 X<> d
 33 ARCL L
 34 CF 24
 35 DSE T
 36 GTO 03
 37 XEQ "VA"
 38 TONE 6
 39 CLA
 40 4
 41 RDN
 42 SF 24
 43 LBL 03
 44 X<> d
 45 RTN
 46 LBL "VA"
 47 SF 25
 48 PRA
 49 FS?C 21
 50 CF 25
 51 AVIEW
 52 FC?C 25
 53 SF 21
 54 END
```
(129 bytes)

6-6 SYNTHETIC FLAG 55 TOGGLE ("55"): This routine will toggle Flag 55 (the Printer Existence Flag) if the printer is not plugged in. If Flag 55 is set, VIEW and AVIEW will not stop program execution, thus allowing the ROM routines to be executed as subroutines of a main program. The stack is left unchanged; it does not change the status of any flag except Flag 55. "55" uses no data registers; the Alpha Register is used, then cleared. Execution time is less than 1 second. To use, simply XEQ "55"; if Flag 55 was clear, it will be set; if it was set, it will be cleared. Source: Valentin Albillo (4747). See 2-31, 3-12, 4-1, 4-18, 6-4, 6-14, 22-23.

01 LBL "55"	04 CLX	07 "├*****"	10 FC?C 15	13 STO M	16 STO d	19 RTN
02 CLA	05 RCL d	08 X<> M	11 SF 15	14 "├**"	17 X<> O	
03 STO N	06 STO M	09 STO d	12 X<> d	15 X<> N	18 CLA	(41 bytes)

6-7 SYNTHETIC RESET FLAGS ("RF"): This routine resets flags to "MASTER CLEAR" status, except that the display mode is set to FIX 2, not FIX 4. Note that USER Mode is turned off (CF 27) and Flag 55 is unaltered. Alpha Register is cleared. Flags set by this routine: 26, 28, 29, 38, 40. Line 02 is nonstandard; it is decimal 244, 44, 2, 128, 0. Source: Valentin Albillo (4747) (PPC ROM).

LBL "RF", ",x̄◆", ASTO d, CF 03, CLA, RTN (17 bytes)

6-8 SYNTHETIC MASS FLAG CONTROL: A synthetic text line of up to seven characters, followed by a 'RCL M', will place an NNN (Non Normalized Number) into the X Register. An important use of NNNs so created is for 'mass flag control' through storage of the NNN into Register d, the Flag Register, allowing the setting or clearing of all 56 flags in one operation. The basic sequence is "XXXXXXX", RCL M, STO d, where "XXXXXXX" represents the synthetic text line used to generate the NNN. This routine uses 12 bytes, the same as would be required for six 'SF nn' or 'CF nn' program lines; hence, use of this routine will save bytes whenever more than six flags are to be set or cleared.

To determine the synthetic text line required to generate the desired flag status, write out the states of all of the flags as a 56-bit binary number, with 1's for set flags and 0's for clear flags, then group the bits into eight-bit hexadecimal bytes. The example below sets Flags 1, 2, 3, 26 (audio enable), 28 (radix), 29 (separator); for 'FIX/ENG 3' display format, it sets Flags 38, 39, 40 & 41; for RAD Mode, it sets Flag 43; for continuous ON, it sets Flag 44; all other flags are clear. Flag 00 is on the left; Flag 55 is on the right.

```
0111 0000   0000 0000   0000 0000   0010 1100   0000 0011   1101 1000   0000 0000
  7    0      0    0      0    0      2    C      0    3      D    8      0    0
```

The required text line, preceded by a TEXT 7 byte, is 'F7 70 00 00 2C 03 D8 00'; the decimal equivalent is 247, 112, 0, 0, 44, 3, 216, 0.

Source: William Wickes (3735) ('Synthetic Programming on the HP-41C'). See 25-6.

6-9 ASSIGNING "TONE" TO THE TOP TWO ROWS OF KEYS: If you assign "TONE" to each of the keys in the top two rows, a double press of each key in USER mode will execute tones 1-5 (top row) and tones 6-0 (second row). Source: George Donaldson (3825) (PPC J, V6N5P19). This same idea applies to other functions; assign FIX to the 1/X key to easily set FIX 2, for example. Synthetics can be used to reduce these useful assignments to a single keystroke. (See the Key Assignments Program ["KA"] in 'Synthetic Programming on the HP-41C', pp 45-47, by William Wickes (3735).

6-10 SYNTHETIC TONE ROUTINES ("T1" - "T5"): "T1" Phasers; "T2" BEEP 2 (no synthetics), "T3" Bach Toccata; "T4" Shave & a Haircut 2 Bits; "T5" Alarm; "T6" Close Encounters. Numbers in parentheses below are the synthetic tone numbers. Source: "T1", Cary Reinstein (2046); "T3", Nicholas Peros (2392); others, Gary Tenzer (1816) (PPC CJ, V7N2P49). Key synthetic tones using the Load Bytes Program; each syn. tone is 2 bytes; the 1st is always 159; the 2nd is any number from 0 to 127.

```
01 LBL "T1"
02 TONE 7 (57)
03 TONE 7 (57)
04 TONE 7 (57)
05 TONE 9 (89)
06 TONE 9 (89)
07 TONE 9 (89)
08 TONE 7 (57)
09 TONE 9 (89)
10 TONE 9 (89)
11 TONE 7 (57)
12 TONE 9 (89)
13 TONE 9 (89)
14 TONE 9 (89)
15 RTN
16 LBL "T2"
17 TONE 1
18 TONE 5
19 TONE 3
20 TONE 6
21 TONE 4
22 TONE 8
23 TONE 9
24 RTN
25 LBL "T3"
26 TONE 1
27 TONE 0
28 TONE 3 (33)
29 XEQ 01
30 XEQ 01
31 XEQ 01
32 TONE 0
33 TONE 5 (15)
34 TONE 4 (94)
35 TONE H (109)
36 TONE b (124)
37 TONE 3 (13)
38 RTN
39 LBL 01
40 X<>  X
41 X<>  X
42 RTN
43 LBL "T4"
44 TONE X (115)
45 TONE 1
46 TONE 1
47 TONE 8 (98)
48 TONE Z (113)
49 TONE b (124)
50 TONE 3 (83)
51 RTN
52 LBL "T5"
53 XEQ 02
54 XEQ 02
55 X<>Y
56 X<>Y
57 X<>Y
58 X<>Y
59 LBL 02
60 TONE 7 (57)
61 TONE 2 (72)
62 TONE 7 (57)
63 TONE 2 (72)
64 TONE 7 (57)
65 TONE 2 (72)
66 TONE 7 (57)
67 TONE 2 (72)
68 RTN
69 TONE "T6"
70 TONE 0
71 TONE 1
72 TONE 5 (15)
73 TONE b (124)
74 TONE 4 (94)
75 END
```

(168 bytes)

6-11 SYNTHETIC MOZART ("MOZ" & "MO"): "MOZ" plays a phrase from "Eine Kleine Nachtmusik", accompanied by an entertaining display; the stack is used. "MO" is a variation with no display that does not change the stack; it can be used as a BEEP alternative. Source: Robert Swanson (5993).

```
01 LBL "MOZ"
02 CLST
03 CF 21
04 "><>MOZART<><"
05 SF 25
06 AVIEW
07 SF 99
08 LBL 01
09 TONE 6 (96)
10 +
11 LBL 01
12 TONE H (109)
13 LBL 01
14 TONE 6 (96)
15 +
16 LBL 01
17 TONE H (109)
18 LBL 01
19 TONE 0 (0)
20 LBL 01
21 TONE H (109)
22 LBL 01
23 TONE 0 (0)
24 LBL 01
25 TONE 2 (2)
26 LBL 01
27 TONE 3 (3)
28 +
29 LBL 01
30 TONE 6 (66)
31 LBL 01
32 TONE 0 (80)
33 TONE 0 (80)
34 LBL 01
35 TONE 6 (66)
36 TONE 3 (83)
37 "***MOZART***"
38 PROMPT
39 GTO "MOZ"
40 END
```

(93 bytes)

```
01 LBL "MO"
02 TONE 6 (96)
03 LBL 01
04 LBL 01
05 TONE H (109)
06 LBL 01
07 TONE 6 (96)
08 LBL 01
09 LBL 01
10 TONE H (109)
11 LBL 01
12 TONE 0 (0)
13 LBL 01
14 TONE H (109)
15 LBL 01
16 TONE 0 (0)
17 LBL 01
18 TONE 2 (2)
19 LBL 01
20 TONE 3 (3)
21 LBL 01
22 LBL 01
23 TONE 6 (66)
24 LBL 01
25 TONE 0 (80)
26 TONE 0 (80)
27 LBL 01
28 TONE 6 (66)
29 TONE 3 (83)
30 END
```

(51 bytes)

6-12 MARY HAD A LITTLE LAMB: This is an amusement routine; it requires SIZE 012. XEQ "MARY"; when the initialization is complete, press R/S as often as desired. If printer is plugged in, CF 21 before executing Mary. Source: Bill Kolb (265) (PPC CJ, V7N1P13 & V7N4P13).

```
01 LBL "MARY"
02 "HOLD ON..."
03 AVIEW
04 " MARY"
05 ASTO 00
06 " HAD"
07 ASTO 01
08 "  A"
09 ASTO 02
10 "LITTLE"
11 ASTO 03
12 " LAMB"
13 ASTO 04
14 " ITS"
15 ASTO 05
16 "FLEECE"
17 ASTO 06
18 " WAS"
19 ASTO 07
20 "WHITE"
21 ASTO 08
22 "  AS"
23 ASTO 09
24 " SNOW"
25 ASTO 10
26 " "
27 ASTO 11
28 "OK-PRESS R/S"
29 PROMPT
30 LBL 00
31 XEQ 01
32 XEQ 02
33 VIEW 03
34 TONE 1
35 TONE 1
36 VIEW 04
37 TONE 1
38 XEQ 02
39 VIEW 03
40 TONE 2
41 TONE 3
42 VIEW 04
43 TONE 3
44 XEQ 02
45 XEQ 01
46 X<>Y
47 VIEW 05
48 TONE 2
```

[continued]

49 VIEW 06	55 VIEW 09	61 GTO 00	67 TONE 0	73 VIEW 04	79 X<>Y
50 TONE 1	56 TONE 1	62 LBL 01	68 VIEW 02	74 TONE 2	80 X<>Y
51 VIEW 07	57 VIEW 10	63 VIEW 00	69 TONE 1	75 RTN	81 X<>Y
52 TONE 1	58 TONE 0	64 TONE 2	70 VIEW 03	76 LBL 02	82 X<>Y
53 VIEW 08	59 VIEW 11	65 TONE 1	71 TONE 2	77 X<>Y	83 END
54 TONE 2	60 STOP	66 VIEW 01	72 TONE 2	78 X<>Y	(224 bytes)

6-13 RESET FLAG 12 WHEN NEEDED: Flag 12, if set, instructs the HP 82143A Printer to print double wide. This flag is cleared when the 41C/V is turned off. For this reason, it is a good practice to set Flag 12 whenever double-wide printing is desired, rather than only once at the beginning of the program. If a program is stopped and the calculator turned off, or the machine 'times out' and turns off, the output may not be as expected when the calculator is turned on and program execution is resumed. Source: HP KEY NOTES, V5N1P7.

6-14 FLAG 21: Flag 21 gives the user control of the printer and its automatic response to VIEW and AVIEW instructions. Flag 21 is automatically set when Flag 55 is set (except when Flag 55 is set synthetically--see 6-4 & 6-6). Flag 55 is set whenever the printer is plugged in. If the printer is not plugged in when the HP-41 is turned on, Flags 55 and 21 are cleared. You do not have any control over Flag 55 (except synthetically), but you may set and clear Flag 21 as desired.

FLAG 21 & VIEW/AVIEW

FLAG 21 SET: Execution stops on (A)VIEW unless printing can occur (printer plugged in & on). Execution stops on (A)VIEW both when printer is not plugged in and when it is plugged in, but is off.

FLAG 21 CLEAR: (A)VIEW never prints and never halts execution.

Perhaps the most confusing situation arises when AVIEW is used in a program and the printer is connected but not turned on. When this occurs, program execution stops at the AVIEW instruction. A solution is to turn on the printer and press R/S. Other solutions: (1) CF 21, R/S; or (2) turn HP-41 OFF, unplug printer, turn HP-41 ON and R/S. The use of Flags 21 and 55 must be carefully planned and tested if the desired combinations of display, printed outputs, or both are to be obtained. Source: HP KEY NOTES, V5N1P7. See 4-1, 4-18, 6-4, 6-6, 2-31, 3-12, 22-23.

6-15 SYNTHETIC SET OR CLEAR ANY FLAG ("SET" & "CLR"): These routines call on the Synthetic Invert Flag Routine, "IF" (6-4).

LBL "SET", FC? IND X, XEQ "IF", RTN
LBL "CLR", FS? IND X, XEQ "IF", RTN

CHAPTER VII

STACK OPERATIONS

7-1 OPERATIONS OF X ON X: Source: Valentin Albillo (4747).
ST+ X: Doubles X. Faster than '2, *'; doesn't disturb Y, Z, T, or L.
ST- X: Similar to CLX; doesn't disable stack lift. 2 bytes rather than 1.
ST* X: Similar to X↑2, but 2 bytes. Doesn't disturb LASTX (L) Register.
ST/ X: Replaces value in X Register with '1'. Doesn't disturb Y, Z, T, or L.

7-2 MULTIPLY (OR DIVIDE) VALUE IN X BY A CONSTANT ONLY IF FLAG CLEAR: Examples:
Good: ..., FS? 00, GTO 14, 5 * (or /), LBL 14, ...
Better: ..., 5, FC? 00, * (or /), FS? 00, RDN, ...
Best: ..., 5, FS? 00, SIGN, * (or /), ... (Non-negative constants only)

7-3 (X,Y) → (X-Y, Y): Put x-y in the X Register, while leaving y in the Y Register. Source: Joseph Horn (1537) (PPC CJ, V7N4P13).

All calculators with "%CH": %CH, %. Rounding errors possible.
HP-41C/V alternative: RCL Y, -. T is lost.

7-4 RCL X, Y, Z OR T: All four are 2-byte instructions.

RCL X: XYZT → XXYZ Similar to ENTER, but the stack lift is not disabled.
ENTER: XYZT → XXYZ.
RCL Y: XYZT → YXYZ Similar to X<>Y, but T is lost and a copy of Y is left in Z.
X<>Y: XYZT → YXZT.
'RCL Y, +' would change XYZT to X+Y, Y, Z, Z.
RCL Z: XYZT → ZXYZ. T is lost.
RCL T: XYZT → TXYZ. Same as the 1-byte instruction R1.

7-5 TO CHANGE THE VALUE IN X TO 1 WITHOUT RAISING THE STACK: ST/ X works for any number except zero. SIGN, ABS changes any number to 1 in the same number of bytes, but replaces the value in L (LASTX Register) with the value in X prior to the operation. Source: PPC Melbourne Chapter.

7-6 DIVIDE X & Y BY 10: Old: 10, /, X<>Y, LASTX, /, X<>Y. New: 10, ST/ Z, /.

7-7 STACK ANALYSIS ("SA"): This routine can be used to test another routine's effect on the stack. XEQ "SA" first, then key values as required and execute the routine to be tested. When execution stops, review stack and L Registers to see which original stack values remain, and where. (29 bytes)

LBL "SA", "L", ASTO L, "T", ASTO T, "Z", ASTO Z, "Y", ASTO Y, "X", ASTO X, END.

7-8 AUTOMATIC STACK REVIEW WITHOUT PRINTER ("ST"): If the printer is plugged in but is turned off, CF 21 first. XEQ "ST" at any time to review the contents of the stack, including L (LASTX Register). Uses the Alpha Register. Source: Bruce Clark (5795).

01 LBL "ST"	04 AVIEW	07 ARCL X	10 "Y= "	13 PSE	16 AVIEW	19 ARCL T	22 CLD
02 "L= "	05 PSE	08 AVIEW	11 ARCL Y	14 "Z= "	17 PSE	20 AVIEW	23 END
03 ARCL L	06 "X= "	09 PSE	12 AVIEW	15 ARCL Z	18 "T= "	21 PSE	(50 bytes)

7-9 STACK EXCHANGE, SAVE & RECALL ("STX", "STS", & "STR"): "STX" (Stack Exchange) exchanges the contents of the L, X, Y, Z & T Registers with the contents of Registers 00, 01, 02, 03 & 04, respectively. "STS" (Stack Save) places a copy of the stack into Registers 00-04. "STR" (Stack Recall) places a copy of the contents of Registers 00-04 into L, X, Y, Z & T respectively. Source: Bill Carter (2998) (PPC CJ V7N7P15).

01 LBL "STX"	07 X<> 02	13 RTN	19 RDN	25 RDN	31 RCL 03
02 X<> L	08 RDN	14 LBL "STS"	20 STO 02	26 RTN	32 RCL 02
03 X<> 00	09 X<> 03	15 X<> L	21 RDN	27 LBL "STR"	33 RCL 01
04 X<> L	10 RDN	16 STO 00	22 STO 03	28 RCL 00	34 END
05 X<> 01	11 X<> 04	17 X<> L	23 RDN	29 STO L	
06 RDN	12 RDN	18 STO 01	24 STO 04	30 RCL 04	(64 bytes)

7-10 INDIRECT STACK SAVE & RECALL ("SM" & "MS"): "SM" (Stack to Memory) stores the stack (X, Y, Z, T & L) in the 5-register block pointed to by the value in R00. Execution of "SM" saves L, but the rest of the stack is lost (recover it with "MS", following). "MS" (Memory to Stack) recalls the contents of the 5-register block pointed to by the value in R00 into the stack (in X, Y, Z, T & L order). It can be used to recall a previously-saved stack. Source: PPC CJ, V7N10P7 (PPC ROM).

01 LBL "SM"	07 STO IND 00	13 RDN	19 4	25 DSE 00	31 RCL IND 00
02 XEQ c	08 4	14 1	20 ST+ 00	26 RCL IND 00	32 END
03 XEQ c	09 ST- 00	15 ST+ 00	21 RCL IND 00	27 DSE 00	
04 XEQ c	10 RTN	16 RDN	22 SIGN	28 RCL IND 00	
05 XEQ c	11 LBL c	17 RTN	23 DSE 00	29 DSE 00	
06 LASTX	12 STO IND 00	18 LBL "MS"	24 RCL IND 00	30 STO X	(68 bytes)

7-11 STACK MANIPULATIONS ("STACK"): This routine can be used to determine the effect on the stack (X, Y, Z & T Registers) of various combinations of stack-manipulating functions, such as X<>Y, RCL T, STO Z, and RDN. XEQ "STACK" to display "X-Y-Z-T" and to put "X" in the X Register, "Y" in the Y Register, and so on. Then perform the stack-manipulating function(s); next, press R/S. The resulting stack arrangement will be shown ("YXTZ" for example, after an X<>Y). For a new case, press R/S. CF 21 before execution if a printer is plugged in. It may be helpful to assign X<> and R↑ to convenient keys. You can even speed execution of X<>Y and RDN by assigning them to their own keys. Source: John Dearing (2791).

01 LBL "STACK"	05 ASTO Y	09 ASTO T	13 ARCL X	17 AVIEW
02 "X"	06 "Z"	10 "X-Y-Z-T"	14 ARCL Y	18 END
03 ASTO X	07 ASTO Z	11 PROMPT	15 ARCL Z	
04 "Y"	08 "T"	12 CLA	16 ARCL T	(47 bytes)

Variation 1: To have routine pause to display the result of stack rearrangements, then turn the calculator OFF, then continue execution when calculator is turned ON again, insert the following steps after step 17 (AVIEW) above: PSE, PSE, SF 11, OFF, GTO "STACK". The routine will now be 59 bytes.

Variation 2: To be able to see the effect on the stack of each of two or more operations, replace step 17 (AVIEW) in the original version above with PROMPT, GTO 00; insert LBL 00 after step 11 (PROMPT); and either assign "STACK" to E (LN) or insert LBL E after step 01. Set USER mode, then (1) XEQ "STACK" [press E]; (2) perform operation(s) on stack; (3) R/S to see stack; and (4) go to step 2 for another operation on the stack as it now exists, or go to step 1 to reset stack to XYZT.

Use the keystroke sequences below in a program to rearrange the stack as desired. For a more complete Stack Manipulation Table, see Reference. The functions that can be used to manipulate the stack include R↑, RDN, ENTER, X<>Y, X<> Z, X<> T, X<> L, STO Y, STO Z, STO T, STO L, RCL X, RCL Y, RCL Z, RCL T, and RCL L. There are several ways to get most stack arrangements; the best is usually the one that takes the fewest bytes. R↑, for example, is 1 byte, while RCL T is 2 bytes.

The symbol "-" below stands for 'exchange'(X-Y for example means X⇌Y or X<>Y).

XYZT	(orig. order)	YXZT	X-Y	ZXYT	X-Y, X-Z	TXYZ	R↑
XYTZ	X-Z, RDN, X-Y	YXTZ	X-Z, RDN	ZXTY	X-Y, RDN, X-Y	TXZY	X-Y, X-T
XZYT	RDN, X-Y, R↑	YZXT	X-Z, X-Y	ZYXT	X-Z	TYXZ	X-Y, R↑
XZTY	X-Y, RDN	YZTX	RDN	ZYTX	RDN, X-Y	TYZX	X-T
XTYZ	R↑, X-Y	YTXZ	RDN, X-Y, RDN	ZTXY	RDN, RDN	TZXY	RDN,RDN, X-Y
XTZY	RDN, RDN, X-Z	YTZX	X-T, X-Y	ZTYX	X-Y, RDN, RDN	TZYX	RDN, X-Z

CHAPTER VIII

MEMORY & CURTAIN

8-1 SYNTHETIC CURTAIN UP ("CU"): This routine takes an integer 'n' in X and adds it to the absolute address of R00 in Status Register c; if 'n' is positive, data registers R00 - R(n-1) will be 'transformed' into program registers, by raising the imaginary 'curtain' separating data and program memory from the original position below R00 to a new position below R(n); R(n) becomes the new R00. If 'n' is negative, the curtain is lowered, so that 'n' registers of program memory are transformed into data registers. All of this occurs without alteration or moving of the contents of the registers involved. It is desirable for programs to use data registers R00 - R15 to save bytes, so it is common to have several programs in memory which use the same block of data registers, and so execution of one program may destroy data used or produced by another. "CU" solves this problem.

To use, key 'n', XEQ "CU". If 'n' is positive, R(n) will become the new R00 (curtain up). If 'n' is negative, R(-n) will become the new R00 (curtain down). All other data registers shift accordingly.

Example: Suppose 'Program 1' is executed, leaving data in R00 - R50 that is required for future use, but in the meantime 'Program 2', using R00 - R25, needs to be run. Key '51', XEQ "CU" (with SIZE 077 or greater), then run 'Program 2'. To restore the curtain to its original position and prepare for a second run of 'Program 1', key '-51', XEQ "CU". **WARNING**: Raising the curtain above the top of memory (i.e., executing "CU" for 'n' greater than the current SIZE), or lowering it below the bottom of memory (below hex '0C0') will cause "MEMORY LOST".

The 41C/V will operate quite normally while the curtain is raised or lowered from the position last established by a SIZE operation. However, if the curtain is raised, changing data into program memory, the memory should not be PACKed, since that will most likely change the data stored below the curtain irreversibly by removing all the null bytes in the data. This difficulty can be avoided if an "END" is placed at the top of program memory, followed by execution of a "PACK". If the curtain is subsequently lowered, the data registers transformed to program memory will be unaffected by the "PACK": they are protected by the "END", which was coded to indicate a packed file. Source: William Wickes (3735) ('Synthetic Programming on the HP-41C') (PPC ROM).

01 LBL "CU"	09 X<> d	17 2	25 FC?C IND Y	33 DSE Y	41 STO M
02 ABS	10 STO O	18 /	26 SF IND Y	34 GTO 01	42 "├ABC"
03 RDN	11 LBL 00	19 RCL M	27 FC? IND Y	35 LBL 13	43 X<> N
04 RCL c	12 RDN	20 X<>Y	28 CHS	36 DSE M	44 X<> c
05 STO M	13 X<> L	21 FRC	29 X>0?	37 GTO 00	45 RDN
06 "├◆◆◆◆"	14 INT	22 X=0?	30 GTO 13	38 LBL 14	46 CLA
07 11	15 X=0?	23 GTO 13	31 FC? IND Y	39 X<> O	47 END
08 X<> M	16 GTO 14	24 LBL 01	32 CHS	40 X<> d	(87 bytes)

8-2 PROGRAM CLEARING RESTRICTIONS: When you wish to clear a very long program (longer than 233 lines), you must set the printer (if present) to MAN (Manual) Mode while executing the "CLP" function. Programs longer than 1089 lines must be cleared using the "DEL" function. For example, to clear a 1980-line program, execute "DEL", then press 'EEX 980'. The END will remain. Source: HP KEY NOTES, V4N1P3.

8-3 REGISTERS REMAINING WHILE PROGRAMMING: After adding an instruction at the end of program memory, you can determine how many memory registers remain unused by pressing "SST". The display will show .END. REG followed by the number of unused registers. Pressing SST again will set the pointer to Line 01 of the current program; pressing BST instead will set the pointer back to the last line of the program, enabling you to continue adding instructions. After inserting an instruction in any program, you can determine how many registers remain completely unused by pressing SHIFT GTO . 000. The display will then be 00 REG followed by the number of unused registers (and the pointer will be at step 00 of the program). Source: 'HP-41C Operating Manual', © Copyright (June 1980) Hewlett-Packard Company. Reproduced with permission.

8-4 SYNTHETIC GETTING TO THE .END. ("EN"): Usually, a program under development is the last program file in memory; i.e., the file containing the ".END.". If the address pointer is moved to some other file, there are only two ways to return it to the last file: use "GTO" and spell out a global label within the program (if there is one), or use "CAT 1", running to the end of the catalog (slow with many programs in the calculator). This routine provides a third method: XEQ "EN"; the program pointer will be set to the top of the program file containing the .END. (SST in PRGM Mode to see Step 01, or BST to resume programming). Source: William Wickes (3735) ('Synthetic Programming on the HP-41C').

01 LBL "EN"	04 "├◆◆◆◆◆"	07 CF 00	10 SF 03	13 STO M	16 STO b
02 RCL c	05 X<> M	08 CF 01	11 X<> d	14 "├◆◆"	17 END
03 STO M	06 X<> d	09 SF 02	12 CLA	15 X<> N	(45 bytes)

8-5 SYNTHETIC CURTAIN FINDER ("C?"): XEQ "C?" to find the curtain location (the absolute address of R00). Line 26 is decimal 244, 127, 0, 0, 65. Source: Keith Jarett (4360) & Roger Hill (4940) (PPC CJ, V7N10P15; PPC ROM). See 1-7.

01 LBL "C?"	07 CF 01	13 FS?C 11	19 FS?C 14	25 X<> d	
02 RCL c	08 CF 02	14 SF 09	20 SF 13	26 E38	
03 STO M	09 CF 04	15 FS?C 12	21 FS?C 15	27 /	
04 "├◆◆A"	10 CF 07	16 SF 10	22 SF 14	28 INT	
05 X<> M	11 FS?C 10	17 FS?C 13	23 FS?C 16	29 DEC	
06 X<> d	12 SF 07	18 SF 11	24 SF 15	30 END	(66 bytes)

8-6 SYNTHETIC HIDE & UNCOVER DATA REGISTERS; ΣREG-CURTAIN EXCHANGE ("HD", "UD" & "ΣC"): Minimum SIZE = k+6. Key k, XEQ "HD" to raise the curtain k registers. An alpha constant is put in the former Register k (now R00). To restore the former curtain location, XEQ "UD". This is an automatic return (using R00) to where you were when "HD" was executed. After "UD" is executed, the old R00 (now Register k) still contains the alpha constant. ["HD" uses the ΣREG-Curtain Exchange Routine ("ΣC"); an example of "ΣC" is: SIZE 010, ΣREG 03, XEQ "ΣC", XEQ "S?" (see 7), XEQ "Σ?" (see -3); XEQ "ΣC", XEQ "S?" (see 10), XEQ "Σ?" (see 3) (see routine 1-17)]. The example below left shows the effect of executing "HD" with k = 5 and initial SIZE = S = 08. After the curtain has been raised, R05 becomes R00 (and its value is replaced with the alpha constant), R06 becomes R01, etc., and the values in old R00-R04 are 'hidden'. Executing "UD" restores the curtain to its former location, but the alpha constant remains in R05 and ΣREG 01 is set. Source: PPC ROM. Jarett (4360).

BEFORE		AFTER
R07 =	7.000	= R02
R06 =	6.000	= R01
R05 =	control	= R00
R04 =	4.000	
R03 =	3.000	
R02 =	2.000	
R01 =	1.000	
R00 =	0.000	

01 LBL "HD"	10 ASTO IND L	19 STO N	28 "├E"
02 SIGN	11 ΣREG IND L	20 "├B"	29 X<> d
03 RDN	12 LBL "ΣC"	21 STO M	30 SCI IND N
04 RCL c	13 CLA	22 "├CD"	31 X<> d
05 "θ"	14 RCL c	23 X<> O	32 STO O
06 X<> M	15 STO O	24 X<> d	33 RDN
07 "├**"	16 STO M	25 SF 08	34 RCL c
08 STO N	17 "├A"	26 X<> d	35 ΣREG 00
09 RDN	18 CLX	27 X<> N	[continued]

```
36 X<> c
37 STO M
38 "├F"
39 RDN
40 RCL N
41 "├GHI"
42 STO L
43 RDN
44 RCL c
45 STO M
46 X<> O
47 "├J"
48 STO M
49 "├K"
50 X<> L
51 STO M
52 "├L"
53 X<> N
54 X<> c
55 RDN
56 CLA
57 RTN
58 LBL "UD"
59 CLA
60 ARCL 00
61 ASTO c
62 ΣREG 01
63 END
```

(141 bytes)

CHAPTER IX

DATA REGISTERS

9-1 DATA REGISTER LOAD & REVIEW ("LD" & "RV"): To store data, XEQ "LD"; you will be prompted for each register from R01 on up. (Both routines use R00). To review data in registers 01 on up, XEQ "RV". Output will print if possible: press R/S to stop execution (or execution will stop with "NONEXISTENT" when routine attempts to recall a nonexistent register). Source: John Dearing (2791) (PPC CJ, V7N4P7).

01 LBL "LD"	07 "R"	13 GTO 01	19 LBL 02	25 FIX 2	31 END
02 1.4	08 ARCL 00	14 RTN	20 FIX 0	26 SF 29	
03 STO 00	09 "⊦=?"	15 LBL "RV"	21 CF 29	27 ARCL IND 00	
04 FIX 0	10 PROMPT	16 SF 21	22 "R"	28 AVIEW	
05 CF 29	11 STO IND 00	17 1.4	23 ARCL 00	29 ISG 00	
06 LBL 01	12 ISG 00	18 STO 00	24 "⊦="	30 GTO 02	(69 bytes)

9-2 RECALL A REGISTER AND RESET TO 0 OR 1: To reset to 0, use (for example) RCL 01, ST- 01. To reset to 1, use RCL 01, ST/ 01 (doesn't work if register contents is 0). Source: Bill Kolb (265) (PB 67/97). Another way to reset to 0 is 0, X<> 01; similarly, to reset to 1, use 1, X<> 01 (works for any value). Source: PPC Melbourne chapter.

9-3 ZERO-ONE TOGGLE: Use '1, -, ABS'. Example: to toggle the contents of Register 00, use ..., RCL 00, 1, -, ABS, STO 00, RDN, ... Source: Joseph Holmes (3673) (PPC CJ, V7N5P7).

9-4 INDIRECT USE OF XEQ FOR DATA RETRIEVAL ("PHONE"): This example recalls a telephone number when given a name of up to six characters. XEQ "PHONE", input name when prompted, R/S, and see phone number. Source: HP KEY NOTES, V4N1P11.

01 LBL "PHONE"	05 AOFF	09 STOP	13 LBL "BOB"	17 "222-2791"
02 "NAME?"	06 ASTO X	10 LBL "JANET"	14 "753-555-6767"	18 END
03 AON	07 XEQ IND X	11 "533-555-1212"	15 RTN	
04 PROMPT	08 AVIEW	12 RTN	16 LBL "NANCY"	(89 bytes)

9-5 "STO" FOLLOWED BY STORAGE REGISTER ARITHMETIC:

STO nn, ST+ nn: Stores 2X (twice the contents of the X Register) in another register, without altering the stack, in only 2 steps.

STO nn, ST- nn: Clears a register without altering the stack.

STO nn, ST* nn: Stores the square of the value in the X Register in another register without altering the stack.

STO nn, ST/ nn: Resets the contents of a register to 1 without altering the stack. Won't work if the value in X is zero.

9-6 CLEARING HIGHER-NUMBERED DATA REGISTERS: When an application requires using data registers, the most vital information should be kept in the lowest-numbered data registers. The highest-numbered data registers may be used for scratch. The technique of sizing down and then sizing up may be used to clear the highest-numbered registers. Source: Richard Nelson (1).

9-7 PACK & UNPACK REGISTER ("PR" & "UR"): Pack Register ("PR") can be used to store data in packed form in a data register; Unpack Register ("UR") can be used to recall packed data from a data register. The packing scheme is to simply encode data, using a base b representation. Using this technique, it is possible to store several numbers in one register. Both routines assume that Register 10 holds the base b and that Register 11 is pointing to the register that will store the data to be packed or that contains a number to be decoded.

R10: base b R11: register pointer

To store the number 'n' in position 'k' of the register pointed to by R11, key n, ENTER, k, XEQ "PR"; the number 'n' must be in the range 0 - (b-1). "PR" calls "UR" and does not return any useful values in the stack. To recall the number stored in position 'k' of the register pointed to be R11, key k, XEQ "UR". "UR" will return in the X Register a number in the range 0 - (b-1).

TABLE OF POSSIBLE BASES & POSITION NUMBERS

Data Range	Base b	Pos. No.	Data Range	Base b	Pos. No.
0-1	2	1-30	0-20	21	1-7
0-2	3	1-19	0-36	37	1-6
0-3	4	1-15	0-99	100	1-5
0-4	5	1-13	0-214	215	1-4
0-6	7	1-11	0-1413	1414	1-3
0-9	10	1-10	0-99999	100000	1-2
0-13	14	1-8			

The most efficient use may be made of data registers by storing the largest values in the lowest-numbered positions and storing the smallest values in the highest-numbered positions. If your priority is the range of data, start with the Data Range column; if your priority is the number of artificial memories available, start with the Position Number column. In many cases, it will be possible to extend the values in this table.

Example: From the above table it can be seen that when the base b = 21, we may store as many as 7 numbers in one register, provided the numbers are in the range 0-20. Use "PR" to pack the numbers 13, 19, 14, 15, 8, 18, and 16 all in R12. Then use "UR" to recall the numbers. Solution: First store the base '21' in R10 and store the register pointer '12' in R11. Then, to store 13 in position 1, key 13, ENTER, 1, XEQ "PR"; to store 19 in position 2, key 19, ENTER, 2, XEQ "PR". Similarly, key 14, ENTER, 3, XEQ "PR", and so on through 16, ENTER, 7, XEQ "PR".

Now recall R12 and see the number 1,447, 473, 103. The base 21 representation of this number shows the seven numbers as coefficients of powers of 21:

$$16*21^6 + 18*21^5 + 8*21^4 + 15*21^3 + 14*21^2 + 19*21^1 + 13 .$$

To use "UR" to recall the numbers in positions 1-7 (from right to left), key 1, XEQ "UR", see '13'; key 2, XEQ "UR", see '19', and so on. Execution time: "PR", 2 seconds; "UR", 1 second. Source: John Kennedy (918) (PPC ROM).

```
01 LBL "UR"    05 X<>Y         09 ST/ Y    13 MOD          17 X<>Y          21 X<>Y
02 1           06 Y↑X          10 X<>Y     14 RTN          18 ST* Z         22 ST+ IND 11
03 -           07 RCL IND 11   11 INT      15 LBL "PR"     19 *             23 END
04 RCL 10      08 X<>Y         12 RCL 10   16 XEQ "UR"     20 ST- IND 11    (43 bytes)
```

9-8 DOUBLE STORAGE: If you have two numbers, preferably one greater than one and the other less than one, if registers are at a premium, and if program space is available, then both numbers can be stored in one register. For example, '127' and '0.35' can be stored in R00 as 127.35: then, to recover the 127, use 'RCL 00, INT'; to recover the 0.35, use 'RCL 00, FRC'. If both numbers are greater (or less) than one, modify after recalling. Source: John Martellaro (1896) (PPC J, V5N1P16).

9-9 FULL DATA REGISTER EXCHANGE & ARITHMETIC: Applies to both numeric and stack registers. Source: John Dearing (2791).

Exchange any 2 registers (--<>--):

Rule: X<> 1st register, X<> 2nd register, X<> 1st register.

Example 1: Exchange Y & R15: X<>Y, X<> 15, X<>Y.

Example 2: Exchange R13 & R17: X<> 13, X<> 17, X<> 13.

First register STO (or ST+, -, * or /) into second register:

Rule: X<> 1st reg, STO (or ST+, -, * or /) 2nd reg, X<> 1st reg.

Example 3: Add Z into R11: X<> Z, ST+ 11, X<> Z.

Example 4: Subtract R05 from R10: X<> 05, ST- 10, X<> 05.

Any other register ST+ (or ST-, ST*, or ST/) into X:

Rule: X<> other reg, ST+ (or ST-, ST*, or ST/) other reg, X<> other reg.

Example 5: Multiply X by R20: X<> 20, ST* 20, X<> 20.

Example 6: Divide Z into X (divide X by Z): X<> Z, ST/ Z, X<> Z.

9-10 COPY ONE REGISTER INTO ANOTHER, WHOSE ADDRESS IS IN X: One solution is to use RCL 1st register, STO IND Y, RDN. Contents of the T Register is replaced by the contents of the 1st register. Another solution is to X<> 1st reg, STO IND 1st reg, X<> 1st reg. This method doesn't raise the stack. Example: Copy the contents of R04 (which is 7) into R03, whose address is in X: Use either 'RCL 04, STO IND Y, RDN' or 'X <> 04, STO IND 04, X <> 04'.

X	3	7	3
R04	7	3	7
R03	?	7	7

9-11 STORE AND RECALL INDIRECT ("SI" & "RI"): These routines are similar to "STO" and "RCL", but work for all data registers, including R100 and above. For "SI", key in the number to be stored and press ENTER (if necessary), then key in the register number and XEQ "SI". T and L Registers are used. For "RI", key in register number to be recalled, then XEQ "RI". L Register is used.

LBL "SI", SIGN, RDN, STO IND L, RTN, LBL "RI", SIGN, RDN, RCL IND L, END (24 bytes)

CHAPTER X

BLOCK OPERATIONS

10-1 LOAD A BLOCK OF REGISTERS WITH THE SAME VALUE: Put the bbb.eee counter in Y and the value in X. For example, to load R01-R13 with '8', use '1.013, ENTER, 8, LBL 14, STO IND Y, ISG Y, GTO 14', ...

10-2 SELF-LOAD ("SLD"): This routine loads every numeric data register with its own address. It will continue execution until R/S is pressed, or until it runs out of registers (at which time it will stop and display "NONEXISTENT"). "SLD" is useful for testing the operation of many block operations. Source: John Dearing (2791) (PPC CJ, V7N5P7).

LBL "SLD", 0, LBL 00, STO IND X, 1, +, GTO 00, RTN

10-3 INPUT BLOCK ("INBL"): This routine prompts for inputs to all data registers in a block. Have the block defined with a bbb.eee control number in X before execution. Source: John Dearing (2791). See 9-1.

01 LBL "INBL"	05 INT	09 LBL 14	13 "⊢=?"	17 ISG Y	21 RTN
02 FIX 0	06 "R0"	10 "R"	14 PROMPT	18 GTO 14	
03 CF 29	07 X=0?	11 ARCL Y	15 STO IND Z	19 FIX 2	
04 ENTER	08 GTO 13	12 LBL 13	16 RDN	20 SF 29	(45 bytes)

10-4 SYNTHETIC INPUT BLOCK ("INB"): Prompts for inputs to all data registers in the block defined by the bbb.eee control number in X; saves the display mode. For numeric entries only. Source: John Dearing (2791).

01 LBL "INB"	05 X=0?	09 RCL d	13 STO d	17 PROMPT	21 GTO 14
02 ENTER	06 GTO 13	10 FIX 0	14 RDN	18 STO IND Z	22 CLX
03 INT	07 LBL 14	11 CF 29	15 LBL 13	19 RDN	23 END
04 "R0"	08 "R"	12 ARCL Z	16 "⊢=?"	20 ISG Y	(46 bytes)

10-5 IMPROVED SYNTHETIC INPUT BLOCK ("IB"): Have the control number (bbb.eee) defining the block in X before execution; it is returned to X after execution. The display mode is saved. At will, ALPHA Mode may be turned on (before keying Alpha characters) or off (before keying numbers). The routine will stay in the mode selected until it is changed by the user. Source: John Dearing (2791).

01 LBL "IB"	07 LBL 14	13 STO d	19 CF 23	25 RDN	31 AOFF
02 ENTER	08 "R"	14 RDN	20 PROMPT	26 FS? 23	32 END
03 INT	09 RCL d	15 LBL 13	21 FC? 23	27 ASTO IND Y	
04 "R0"	10 FIX 0	16 "⊢=?"	22 STO IND T	28 ISG Y	
05 X=0?	11 CF 29	17 ENTER	23 FC? 23	29 GTO 14	
06 GTO 13	12 ARCL Z	18 ASTO X	24 RDN	30 LASTX	(60 bytes)

10-6 BLOCK INCREMENT ("BI"): Have control number bbb.eee defining the block in Z, the initial value to be stored in the first register of the block in Y, and the increment value (pos. or neg.) in X; then XEQ "BI". Example: '5.00902, ENTER, 50, ENTER, 10, XEQ "BI"' puts 50 in R05, 60 in R07 & 70 in R09. Source: PPC ROM.

01 LBL "BI"	03 LBL 10	05 +	07 ISG Y	09 END	
02 -	04 LASTX	06 STO IND Y	08 GTO 10		(19 bytes)

10-7 VIEW BLOCK ("VB"): This routine views/prints the contents of all nonzero registers in a block. Have the block defined with a bbb.eee control number in X, then XEQ "VB". Uses '0, X=Y?' rather than 'X=0?' to avoid the "ALPHA DATA" message if a register contains an Alpha string. Uses more steps than would otherwise be necessary to allow for viewing of R00. Change steps 28 &/or 35 (FIX 2) to suit. To clear any register while executing "VB" without printer, store 0 in the displayed register, then R/S to continue routine execution. Source: John Dearing (2791). See 9-1.

01 LBL "VB"	08 GTO 10	15 0	22 10	29 ARCL IND L	36 SF 29
02 SF 21	09 RCL L	16 RCL IND L	23 X>Y?	30 AVIEW	37 CLST
03 CF 29	10 INT	17 X=Y?	24 "├0"	31 CLD	38 END
04 STO L	11 "R00"	18 GTO 10	25 ARCL L	32 LBL 10	
05 0	12 X=0?	19 FIX 0	26 LBL 11	33 ISG L	
06 RCL IND L	13 GTO 11	20 "R"	27 "├= "	34 GTO 12	
07 X=Y?	14 LBL 12	21 LASTX	28 FIX 2	35 FIX 2	(72 bytes)

10-8 SYNTHETIC VIEW BLOCK ("VB"): To change the View Block routine (10-7) above to a synthetic version in which the display mode before execution determines the display of the registers, and in which the original display mode is retained, make the following modifications: Delete steps 36 (SF 29), 35 (FIX 2), 28 (FIX 2), and 25 (ARCL L); after step 24 ("├0"), insert 'RCL d, CF 29, FIX 0, ARCL L, STO d'; delete steps 19 (FIX 0) and 03 (CF 29).

10-9 SYNTHETIC BLOCK VIEW ("BV"): Key control number bbb.eee defining the block, then XEQ "BV": the contents of all registers in the block that are nonzero will be AVIEWed (and printed if printer is on and Flag 21 is set). For a longer viewing, SF 09 before execution (pauses after each AVIEW); or SF 10 instead to have the routine stop after each AVIEW. Step 29 is synthetic Tone 38. The original display mode is restored. Source: Richard Schwartz (2289) (PPC ROM).

01 LBL "BV"	09 X<> Z	17 "├: "	25 LASTX	33 RTN	41 FC?C 25
02 .	10 INT	18 R↑	26 .	34 LBL "VA"	42 SF 21
03 ENTER	11 CLA	19 ARCL X	27 ENTER	35 SF 25	43 END
04 LBL 00	12 RCL d	20 XEQ "VA"	28 LBL 01	36 PRA	
05 CLX	13 CF 29	21 FS? 10	29 TONE 8 (38)	37 SF 25	
06 RCL IND Z	14 FIX 0	22 STOP	30 ISG Z	38 FS?C 21	
07 X=Y?	15 ARCL Y	23 FS? 09	31 GTO 00	39 CF 25	
08 GTO 01	16 STO d	24 PSE	32 TONE 6	40 AVIEW	(83 bytes)

10-10 REVERSE BLOCK ("RB"): This routine reverses (inverts) any block of numeric data registers as specified by a 'bbb.eee' control number in X. For instance, with '4.008 in X, execution of "RB" moves the contents of R08 to R04, R07 to R05, R06 is unchanged, R05 to R07, and R04 to R08. Especially useful to change an ascendent sort to descendent, or vice versa. The routine itself uses no numeric data registers, and it is fast: it inverts 100 registers in about 13 seconds. Source: Valentin Albillo (4747).

01 LBL "RB"	05 1 E3	09 2	13 /	17 X<> IND Y	21 DSE Y
02 ENTER	06 *	10 /	14 +	18 X<> IND Z	22 GTO 00
03 ENTER	07 ST+ Y	11 INT	15 X<>Y	19 X<> IND Y	23 END
04 FRC	08 X<>Y	12 1 E3	16 LBL 00	20 ISG Z	(41 bytes)

10-11 DUPLICATE BLOCK ("DUP"): This routine copies the data in one block of data registers into a second block. Overlapping of blocks not permitted if moving data to higher-numbered registers. To use, key control number defining block to be saved (bbb.eee), ENTER, first register of destination block (BBB), XEQ "DUP". Source: Bill Kolb (265) (PPC CJ, V7N4P17).

LBL "DUP", LBL 00, RCL IND Y, STO IND Y, RDN, 1, +, ISG Y, GTO 00, RTN. (20 bytes)

10-12 BLOCK MOVE ("BM"): This routine moves a block of data registers to a new location. In other words, it duplicates, copys or saves the block in a new location without changing the values in the block that is copied. Old values in the destination block are lost. The original or sending block and the destination block may overlap. "BM" uses no numeric data registers of its own. To use, key 1st reg. to be moved, ENTER, 1st reg. of destination block, ENTER, no. of reg. in block, XEQ "BM". Source: John Kennedy (918) (PPC ROM).

```
01 LBL "BM"
02 SIGN
03 RDN
04 X<Y?
05 GTO 04
06 LASTX
07 ST+ Z
08 +
09 -1
10 ST+ Z
11 ST+ Y
12 RDN
13 LBL 04
14 R↑
15 LBL 05
16 RCL IND Z
17 STO IND Z
18 RDN
19 ST+ Z
20 ST+ Y
21 DSE L
22 GTO 05
23 END
```

REQUIRED STACK BEFORE EXECUTION:

T: -
Z: 1st register to be moved
Y: 1st reg. in destination block
X: no. of consecutive reg. to move

(41 bytes)

10-13 EXCHANGE BLOCK ("XB"): This routine will exchange the contents of two equal-length blocks of consecutive data registers. The blocks must not overlap. To use, key the first register number of one block, ENTER, the first register number of the other block, ENTER, the number of registers to exchange, XEQ "XB". With input of 0, ENTER, 10, ENTER, 10, this routine will simulate the primary-secondary register exchange function of the HP-67/97. Source: John Kennedy (918) & Richard Schwartz (2289) (PPC CJ, V7N10P7). (26 bytes)

```
01 LBL "XB"
02 SIGN
03 LBL 02
04 RCL IND Z
05 X<> IND Z
06 STO IND T
07 RDN
08 ST+ Z
09 ST+ Y
10 DSE L
11 GTO 02
12 END
```

REQUIRED STACK BEFORE EXECUTION:

T: -
Z: 1st address of one block
Y: 1st address of other block
X: no. of reg. in either block

10-14 BLOCK EXCHANGE ("BE"): Exchanges the contents of one block with the contents of a second block of equal length. Key begin.end registers (bbb.eee) of one block, ENTER, beginning register (BBB) of second block, XEQ "BE". To add a primary-secondary register exchange function (simulating the HP-67/97 P⇄S function), precede LBL "BE" with 'LBL "P<>S", .009, ENTER, 10'. Source: Bill Kolb (265) (PPC CJ, V7N4 P17; PPC ROM).

```
01 LBL "BE"
02 LBL 01
03 RCL IND Y
04 X<> IND Y
05 STO IND Z
06 RDN
07 ISG X
08 STO X
09 ISG Y
10 GTO 01
11 END
```

(25 bytes)

10-15 SYNTHETIC PRIMARY-SECONDARY EXCHANGE ("P-S"): This routine duplicates the P⇄S function of the HP-67/97 by exchanging the contents of R00-09 with the contents of R10-19. The stack is saved; the Alpha Register is used; minimum SIZE is 20. Execution time, 3.5 seconds. Source: David Bartholomew (3666) (PPC CJ, V7N8P8).

```
01 LBL "P-S"
02 STO M
03 RDN
04 STO N
05 RDN
06 STO O
07 RDN
08 19
09 ENTER
10 9
11 LBL 01
12 RCL IND X
13 X<> IND Z
14 STO IND Y
15 RDN
16 DSE Y
17 DSE X
18 GTO 01
19 RCL 00
20 X<> 10
21 STO 00
22 R↑
23 RCL O
24 RCL N
25 RCL M
26 END
```

(48 bytes)

10-16 BLOCK ROTATE ("BLR"): This routine rotates or shifts the contents of a block of data registers, defined by a control number (bbb.eee) in X. For example, if you input 2.004, then XEQ "BLR", the contents of R02 is moved to R03, the contents of R03 is moved to R04, and the contents of R04 is moved to R02. The values in X & Y before the control number is keyed in are returned to Z & T, respectively. Source: John Kennedy (918) (PPC CJ, V7N10P9).

```
01 LBL "BLR"
02 ENTER
03 ABS
04 RDN
05 RCL IND L
06 LBL 07
07 ISG L
08 GTO 08
09 STO IND Y
10 RTN
11 LBL 08
12 X<> IND L
13 GTO 07
14 END
```

(28 bytes)

10-17 BLOCK ROTATE IN EITHER DIRECTION ("BR"): The input to this routine is the number of the first register of the block in Y and ±n (where 'n' is the number of registers within the block) in X; the sign of n determines the direction of the rotation. If n is positive, values are moved to the next higher-numbered register (and the contents of the highest-numbered register in the block is moved to the lowest); conversely, if n is negative, values are moved to the next lower-numbered register (and the contents of the lowest-numbered register in the block is moved to the highest). Source: John Kennedy (918) & Richard Schwartz (2289) (PPC ROM).

```
01 LBL "BR"
02 CHS
03 X<0?
04 GTO 07
05 RCL Y
06 X<>Y
07 1
08 ST+ Z
09 -
10 SIGN
11 LBL 06
12 RCL IND Z
13 X<> IND Z
14 STO IND T
15 RDN
16 ST+ Z
17 ST+ Y
18 DSE L
19 GTO 06
20 RTN
21 LBL 07
22 CHS
23 1
24 -
25 +
26 STO Y
27 -1
28 ST+ Z
29 GTO 06
30 END
```

(51 bytes)

10-18 SYNTHETIC BLOCK EXTREMES ("BX"): Have the block defined with a bbb.eee control number in X, then XEQ "BX". The smallest value in the block is returned to X and the largest value is returned to Y. If Flag 10 is set, absolute values are used. The original control number is returned to Synthetic Register O, the register number of the smallest value to Register N, and the register number of the largest value to Register M. Contents of the block are undisturbed. Numeric registers can be used in place of the synthetic registers, or steps 37, 32, 04, 03 & 02 can be deleted. This routine may also be considered to be a matrix routine, since it can be used to determine pivoting operations. Source: Richard Schwartz (2289) (PPC ROM).

```
01 LBL "BX"
02 STO M
03 STO N
04 STO O
05 RCL IND X
06 FS? 10
07 ABS
08 ENTER
09 ENTER
10 RDN
11 LBL 08
12 CLX
13 RCL IND Z
14 FS? 10
15 ABS
16 X>Y?
17 GTO 10
18 R↑
19 X>Y?
20 GTO 11
21 RDN
22 LBL 09
23 ISG Z
24 GTO 08
25 X<>Y
26 R↑
27 RTN
28 LBL 10
29 X<>Y
30 CLX
31 RCL Z
32 STO M
33 GTO 09
34 LBL 11
35 CLX
36 RCL T
37 STO N
38 X<>Y
39 RDN
40 GTO 09
41 END
```

(65 bytes)

10-19 MULTI-REGISTER CLEAR ("CLRGX"): With 'nn' in X, this routine will clear Registers 00 - nn. The values that were in X, Y & Z before 'nn' was keyed in are returned to Y, Z & T. Source: John Dearing (2791) (PPC CJ, V7N5P7).

LBL "CLRGX", STO 00, CLX, LBL 14, STO IND 00, DSE 00, GTO 14, RTN (19 bytes)

10-20 ERASE BLOCK ("EB"): To clear any block of data storage registers, put the control number (bbb.eee) defining the block in X, then XEQ "EB". To clear Registers 05 - 09, for example, key '5.009', XEQ "EB". You may wish to add two RDNs at the end to restore X and Y. Source: John Dearing (2791) (PPC CJ, V7N4P22).

LBL "EB", 0, LBL 13, STO IND Y, ISG Y, GTO 13, RTN (15 bytes)

10-21 MULTIPLE-REGISTER CLEAR USING CLΣ: "CLΣ" clears six adjacent data storage registers, beginning with the register specified by the "ΣREG" function. To clear Registers 10 - 18, for example, use 'ΣREG 10, CLΣ, ΣREG 13, CLΣ'. 6 bytes. ΣREG may need to be reset. Source: HP KEY NOTES, V4N1P5.

10-22 CLEAR REGISTERS WITH NO NUMERIC LABELS ("CR"): To clear any contiguous set of data storage registers, put the bbb.eee (begin.end) control number in X, then XEQ "CR". Source: John Burkhart (4382).

LBL "CR", 0, STO IND Y, RDN, ISG X, GTO "CR", RTN (17 bytes)

10-23 BLOCK CLEAR ("BC"): With the block defined by the bbb.eee control number in X, XEQ "BC" to clear all registers in the block. To clear every other register in a block (to clear alternate registers), use the full control number of the form 'bbb.eeeii', with 'ii' = 02. For example, '10.02002, XEQ "BC"' clears Registers 10, 12, 14, 16, 18 & 20. Y, Z & T Registers are not used. SIGN is used in step 02 rather than STO L in order to save one byte. Source: John Kennedy (918) (PPC ROM).

LBL "BC", SIGN, CLX, LBL 13, STO IND L, ISG L, GTO 13, RTN (16 bytes)

10-24 SELECTION WITHOUT REPLACEMENT ("SE"): This routine can be used to select at random an element from any block of consecutive registers. Subsequent items selected from the block will not repeat. It can also be considered to be a random shuffler which will scramble the contents of a block of registers. "SE" calls the random number generating routine "RN". To initialize, store the number of the first register of the block to be selected from in R06; put the number of registers in the block in R07. Store a fractional seed in any convenient register; call this register 'k'. Once initialized, the normal input to "SE" is simply the number 'k' which points to the random decimal register; key 'k' and XEQ "SE". The output from "SE" is the register content chosen at random and is left in the X Register. Each time "SE" is called, the counter in R07 is decreased by one.

If many calls are being made to "SE", then R07 should be tested for zero before "SE" is called. When R07 is zero, all the available items will have been selected; the items remain stored in the original block but will be rearranged. After a complete shuffling, the items are in the reverse order of selection (the last selected is in the lowest-numbered register). To repeat the selection process, reinitialize the number in R07. This routine uses the stack; it uses no flags. It executes in about 1½ seconds. [Steps 16 (RCL IND X) and 22 (STO IND Y) (part of "RN") could be changed to direct operations (say RCL 05 & STO 05)--then the fractional seed must be stored in that register, but 'k' (5 in this case) need not be keyed in before each execution of "SE"].

Example: (1) Load data as shown below left; to let 'k' = 5, store 0.141592654 in R05. Store '10' in R06 (first register of the block) and store '5' in R07 (the number of registers in the block). (2) Key in '5' (k), XEQ "SE"; see "SUSAN"; the block is now as shown below middle. (3) Repeat step 2 four more times for a complete reshuffling; the block is now as shown below right. (4) To repeat selection from this block, reinitialize R07 to 5, then go to step 2. This routine will, of course, work just as well when the block contains numbers.

R10: ROBERT)		R10: ROBERT)		R10: JOE)	
R11: JANE)	R07	R11: JANE)	R07	R11: GEORGE)	R07
R12: JOE)	= 5	R12: JOE)	= 4	R12: JANE)	= 0
R13: SUSAN)		R13: GEORGE)		R13: ROBERT)	
R14: GEORGE)		R14: SUSAN)		R14: SUSAN)	

Source: Bill Kolb (265) & John Kennedy (918) (PPC ROM).

R'k': Fractional Seed R06: 1st reg. of block R07: No. of reg. in block

01 LBL "SE"	05 RCL 06	09 RCL 07	13 STO IND Y	17 9821	21 FRC
02 XEQ "RN"	06 ST+ Y	10 +	14 RTN	18 *	22 STO IND Y
03 RCL 07	07 DSE 07	11 RCL IND X	15 LBL "RN"	19 .211327	23 END
04 *	08 STO X	12 X<> IND Z	16 RCL IND X	20 +	(54 bytes)

10-25 ODD-EVEN REGISTER EXCHANGE ("OE"): This routine exchanges the contents of adjacent registers, as directed by a control number (bbb.eee) in X defining the block. To avoid confusion, note the following: if the beginning register pointed to is odd-numbered and the ending register pointed to is even-numbered (or vice versa), then 'bbb' is the first register whose contents will be changed, and 'eee' is the last register whose contents will be changed. For example, 1.004, XEQ "OE" will exchange R01 with R02 and R03 with R04; 2.005, XEQ "OE" will exchange R02 with R03 and R04 with R05. The values in X and Y before keying the control number are returned to X and Y. Source: John Herzfeld (5428). [continued]

01 LBL "OE"	04 1.4	07 LBL 00	10 STO IND L	13 ISG L	16 END
02 2 E-5	05 X<>Y	08 RCL IND L	11 RDN	14 GTO 00	
03 +	06 +	09 X<> IND Y	12 ISG X	15 RDN	(34 bytes)

10-26 BLOCK REVIEW & EDIT ("B?"): This routine can be used to enter, edit or review data in a block of data registers. Specify a block-defining control number in the form bbb.eee, as in an ISG instruction, where bbb is the first register and eee is the last. (You may also specify an increment size, as in an ISG instruction.) As the routine executes, it will display the register number and the current contents of the register. If you simply hit R/S, the contents are unaltered. If you enter a new number and hit R/S, the new number replaces the old contents of the data register. OPTION: After step 17 (RDN), insert 'FC? 22'; this change will result in a review of the newly-changed register before going to the next one. Y (the value in X before keying in the control number) is returned to X. Source: Larry Trammell (6824).

01 LBL "B?"	05 FIX 0	09 CF 22	13 FS? 22	17 RDN	21 FIX 2
02 LBL 14	06 "R"	10 "⊦ "	14 STO IND Y	18 ISG X	22 END
03 ENTER	07 ARCL X	11 ARCL IND X	15 FS? 22	19 GTO 14	
04 INT	08 SCI 5	12 PROMPT	16 RDN	20 RDN	(43 bytes)

CHAPTER XI

MATRICES & DATA PROCESSING

11-1 MATRIX ROUTINES ("M1" - "M5"): Each of the five matrix routines requires two stored values, one of which is the starting register of the matrix and the other of which is the number of columns in the matrix. Matrices are assumed to be stored with each row occupying a consecutive block of registers. Thus the number of columns is the block size and the entire matrix is stored row by row as one string of consecutive registers. R07 holds the starting register of the matrix, and R08 holds the number of columns. Both row and column numbers start counting from one.

"M1" will interchange two rows in a matrix; input is the numbers of the two rows to be interchanged. ["M1" may also be considered part of the data base management routines INPUT & DELETE RECORD ("IR" & "DR"), as it can be used to interchange two records; input in this case is the two record numbers.]

"M2" will multiply a row in a matrix by a constant; input is the constant and the row number. As part of a data base management system, "M2" can be used to multiply a numerical record by a constant, including 0 (so input is the constant and the record number).

"M3" will add a multiple of one row in a matrix to another row. The row that is added to changes; the row that is multiplied by the constant does not change. "M3" may be considered part of the data base system routines "IR" & "DR": when records consist of numerical entries (such as columns of prices), "M3" may be used to add a multiple of one record to another.

"M4" will determine the (i,j) element in a matrix (row i, column j), given the number of the data register which contains that element. Input to "M4" is the register number. As part of the data base management system, "M4" can be used to determine a particular field element in a record.

"M5" is the inverse of "M4", and will determine the register number of the (i,j) element in a matrix. Input to "M5" is the row number 'i' and the column number 'j'. As part of the "IR"/"DR" data base management system, "M5" can be used to locate a particular field element in a record, given the record number and the number of the desired item within the record.

Sample Matrix: This 6x5 matrix (below left) is assumed to be stored in R15-R44. The element in the upper left corner ('21') is row 1, column 1 (1,1). The registers involved are shown below left.

21	35	55	74	83	:	R15: 21	R21: 93	R27: 32	R33: 62	R39: 82
11	93	56	36	29	:	R16: 35	R22: 56	R28: 27	R34: 97	R40: 23
65	78	32	27	75	:	R17: 55	R23: 36	R29: 75	R35: 54	R41: 45
53	94	46	62	97	:	R18: 74	R24: 29	R30: 53	R36: 39	R42: 77
54	39	61	67	82	:	R19: 83	R25: 65	R31: 94	R37: 61	R43: 15
23	45	77	15	25	:	R20: 11	R26: 78	R32: 46	R38: 67	R44: 25

This matrix starts in R15 and the number of columns in the matrix is 5, so the following must be stored:

R07: 15 = starting register;   R08 = # of columns.

Any number of matrix operations may be performed on the above matrix without changing the numbers in R07 and R08. With the matrix stored as above, and the starting register and number of columns stored in R07 and R08 respectively, as above, the following operations may be performed:

(1) "M1": To interchange any two rows in the matrix, key the two row numbers into Y & X (order unimportant) and XEQ "M1". A block exchange of the two rows occurs. Example: key 2, ENTER, 4, XEQ "M1" to exchange rows two and four in the example above.

(2) "M2": To multiply row 'i' by the constant 'k', key k, ENTER, i, XEQ "M2". Example: to multiply the last row in the example above by 2, key 2, ENTER, 6, XEQ "M2".

(3) "M3": To add k times row i to row j, key j, ENTER, i, ENTER, k, XEQ "M3". Example: to add -2 times row 3 to row 4 in the sample matrix above, key 4, ENTER, 3, ENTER, 2, CHS, XEQ "M3"; row 4 will now be -77, -62, -18, 8, -53.

(4) "M4": To determine the (i,j) address of the matrix element stored in Register 'r', key 'r', XEQ "M4"; the matrix will be left unchanged. The column number (j) will be returned to X, and the row number (i) will be returned to Y. Example: find the row and column numbers of the element stored in R38 in the sample matrix above: key 38, XEQ "M4"; 4 is returned to X and 5 to Y, so the element in R38 is the (5,4) element (in column 4, row 5).

(5) "M5": To determine the register number of the (i,j)element in a matrix, key 'i' (row), ENTER, 'j' (column), XEQ "M5". The register number will be returned to X. Example: to find the register number of the (2,3) element in the sample matrix above, key 2, ENTER, 3, XEQ "M5"; 22 is returned to X [the (2,3) element is in R22]. To check, key RCL IND X and see '56', which is the (2,3) element.

Source: John Kennedy (918) (PPC ROM).

```
01 LBL "M2"
02 XEQ 00
03 X<>Y
04 LBL 01
05 ST* IND Y
06 ISG Y
07 GTO 01
08 RTN
09 LBL "M3"
10 STO M
11 RDN
12 XEQ 00
13 X<>Y
14 XEQ 00
15 RCL M
16 SIGN
17 LBL 02
18 RDN
19 RCL IND Y
20 LASTX
21 *
22 ST+ IND Y
23 ISG Y
24 ""
25 ISG Z
26 GTO 02
27 RTN
28 LBL "M1"
29 XEQ 00
30 X<>Y
31 XEQ 00
32 LBL "BE"
33 RCL IND Y
34 X<> IND Y
35 STO IND Z
36 RDN
37 ISG X
38 ""
39 ISG Y
40 GTO "BE"
41 RTN
42 LBL 00
43 RCL 08
44 *
45 RCL 07
46 +
47 RCL X
48 RCL 08
49 ST- Z
50 SIGN
51 -
52  E3
53 /
54 +
55 RTN
56 LBL "M4"
57 RCL 07
58 -
59 RCL 08
60 XEQ "QR"
61 ISG Y
62 ""
63 ISG X
64 ""
65 RTN
66 LBL "M5"
67 X<> 08
68 ST- 08
69 *
70 ST+ 08
71 X<> L
72 X<> 08
73 1
74 -
75 RCL 07
76 +
77 RTN
78 LBL "QR"
79 X<>Y
80 STO O
81 X<>Y
82 MOD
83 ST- O
84 LASTX
85 ST/ O
86 CLX
87 X<> O
88 X<>Y
89 END
   (171 bytes)
```

For a nonsynthetic version, make the following changes: Change references to synthetic Register M (steps 10 & 15) to a convenient numeric register, say R00. Likewise, change references to Register O (steps 80, 83, 85 & 87) to a numeric register such as R01. Change the Text 0 NOPs (steps 24, 38, 62 & 64) to any nonsynthetic NOP, such as STO X. Finally, change the short-form exponent 'E3' (step 52) to its nonsynthetic equivalent, '1 E3'.

11-2 INSERT & DELETE RECORD ("IR" & "DR"): These routines can be considered to be part of a data base management system; they apply to files consisting of fixed length records where each record is a block of consecutive data registers. The entire file consists of one large block of consecutive registers. "IR" is a special block move routine which makes room between two file records for insertion of a new record; "DR" deletes a given record from the file and moves the remaining files into the vacated space so that the data area is as compact as possible. Before executing "IR" or "DR", have the file in data memory, and have the following three registers loaded as specified for use by either routine:

R07: s = starting register of the entire file
R08: c = number of consecutive registers per record (# of columns)
R09: n = total number of records in the file (# of rows)

To make room to insert a new kth record, key 'k', then XEQ "IR". "IR" will move the records following and including the kth record into higher-numbered registers to make room to insert new data for a new kth record, and will also add 1 to R09 to update the new number of records. Note that this will cause a change in the numbering of the records following and including the old kth record. Note also that new data is not actually inserted; "IR" simply makes room so that the new record can be inserted between previously existing records.

To delete the kth record from the file, key 'k', XEQ "DR". The records following the kth record will be moved into registers occupied by the old kth record and 1 will be subtracted from R09 to update the new number of records. Note that this will cause a change in the numbering of the records following the kth record.

Example: Assume the records of the original file are as follows, and we are to insert a new record #3:

#1: Joe Robinson	#2: Mike Johnson	#3: Jane Hamilton	#4: Paul Jones
354-1662	363-5648	261-2347	745-3254
Gary, IN	Boston, MA	Fresno, CA	Denver, CO

This sample file is stored in R10-R33, where each record consists of six consecutive registers:

R10: JOE	R16: MIKE	R22: JANE	R28: PAUL
R11: ROBINS	R17: JOHNSO	R23: HAMILT	R29: JONES
R12: ON	R18: N	R24: ON	R30:
R13: 354.1662	R19: 363.5648	R25: 261.2347	R31: 745.3254
R14: GARY	R20: BOSTON	R26: FRESNO	R32: DENVER
R15: IN	R21: MA	R27: CA	R33: CO

Put the following information into R07 - R09:

R07: 10 = starting register of file
R08: 6 = no. of registers/record
R09: 4 = total number of records

Next, key '3', XEQ "IR". Now the third and fourth records are moved into higher-numbered registers (R28-R33 & R34-R39), making room in R22-R27 for a new record to be inserted. Also, the record count in R09 is incremented by one to 5.

To delete this newly-entered record in R22-R27 (the new record #3), key '3', XEQ "DR"; the file is restored to its original form and R09 is decremented by one to 4.

Source: John Kennedy (918) (PPC ROM).

```
01 LBL "IR"
02 ISG 09
03 ""
04 XEQ 03
05 ST- T
06 *
07 GTO "BM"
08 LBL 03
09 RCL 07
10 RCL 08
11 RCL Z
12 *
13 +
14 STO Y
15 RCL 09
16 R↑
17 -
18 RCL 08
19 RTN
20 LBL "DR"
21 XEQ 03
22 ST- Z
23 *
24 DSE 09
25 ""
26 LBL "BM"
27 SIGN
28 RDN
29 X<Y?
30 GTO 04
31 LASTX
32 ST+ Z
33 +
34 -1
35 ST+ Z
36 ST+ Y
37 RDN
38 LBL 04
39 R↑
40 LBL 05
41 RCL IND Z
42 STO IND Z
43 RDN
44 ST+ Z
45 ST+ Y
46 DSE L
47 GTO 05
48 END
```

(89 bytes)

Synthetic NOPs above (steps 03 & 25) can be replaced with "STO X".

11-3 MATRIX INPUT/OUTPUT ("MIO"): This routine prompts for the input to a matrix (numeric input only), using a (row, column) prompt; it is consistent with the matrix routines "M1"-"M5" (11-1). Matrices are stored row by row with each row occupying a block of consecutive registers; the entire matrix is stored as one large block of consecutive registers. To achieve maximum size, store the matrix in R10 on up. This routine calls on routines "M1"-"M5", which in turn call on routine "QR" (all in 11-1); these routines must also be in memory. To get into the "MIO" routine so that keys 'A', 'B' and 'C' are active, press its assigned key if any, or else XEQ "MIO", GTO "MIO", or do a CAT 1 and stop at LBL "MIO"; once in the program, the following functions are available, in USER Mode:

FLAGS:
09: used
29: cleared

REGISTERS:
R03: counter
R04: pointer
R05: # digits displayed
R07: start register
R08: # columns
R09: # rows

A: Input New Matrix	B: Review Matrix	C: Recall (y,x)

Example 1: Input the required matrix, preparatory to solving the following system of equations (see "RRM" [11-4]).

$$\begin{aligned} -5x + 10y + 15z &= 5 \\ 2x + y + z &= 6 \\ x + 3y - 2z &= 13 \end{aligned} \quad = \quad \left[\begin{array}{ccc:c} -5 & 10 & 15 & 5 \\ 2 & 1 & 1 & 6 \\ 1 & 3 & -2 & 13 \end{array}\right]$$

Solution: Get into "MIO", press 'A' for inputting a new matrix; as prompted, input '10', R/S for starting register (if printer is plugged in but OFF, CF 21 first), then '3', ENTER, '4' for (row, column) matrix dimensions. Next, enter one by one the coeffieients (left to right, top to bottom), following each with R/S; the routine will sound a tone and prompt for each coefficient: for example, see "(1,1)=?", key '5', CHS, R/S. Do the same for '10', '15', '5', '2', and so on. After keying the last coeffieient ['13' here--element (3,4)], "BEEP" will sound. To verify the data input, store a number (say '4') in R05 for the number of decimal places to display, then press 'B' to run through the entire matrix; if printer is ON, the matrix will print. If scientific notation is preferred, change line 44 from FIX IND 05 to SCI IND 05. (To have routine stop with each display: if no printer is plugged in, SF 21 1st; if printer is plugged in but is OFF, SF 21 if necessary first.) The first display is typical: "(1,1) = -5.0000". Routine BEEPs after last element is displayed/printed.

Key 'C' may be used to find out which register a particular element is in. Key the row and column numbers of the matrix element you wish to view, then press 'C'. For example, to verify that the (3,2) element is '3', key 3, ENTER, 2, press 'C'; see 'R19.0000", then "(3,2) = 3.0000". Thus, the (3,2) element is stored in R19, and is '3'. Note: if an incorrect entry is made during the automatic input phase (using key 'A'), simply continue entering elements as directed by the display; after all entries have been made, use 'C' to determine which register to manually store the correct element in.

Example 2: Input the required matrix for the following systems of equations, preparatory to solving the equations, finding the inverse of the coeffieient matrix, and finding the determinant (see 11-4):

$$\begin{aligned} 14x + 2y - 6z &= 9 \\ -4x + y + 9z &= 3 \\ 6x - 4y + 3z &= -4 \end{aligned} \quad = \quad \left[\begin{array}{ccc:ccc:c} 14 & 2 & -6 & 1 & 0 & 0 & 9 \\ -4 & 1 & 9 & 0 & 1 & 0 & 3 \\ 6 & -4 & 3 & 0 & 0 & 1 & -4 \end{array}\right]$$

The matrix to be entered will consist of the original coefficient matrix augmented by the identity matrix and augmented by the final column of constants; this is a 3x7 matrix, as shown above right.

Solution: In the "MIO" routine, with USER Mode on, press 'A'; input '10' for starting register and 3, ENTER, 7 for dimensions; then as prompted enter all elements of the above matrix [(1,1) = '14', (1,2) = '2', ..., (3,7) = '-4']. Review with 'B'; if necessary, determine which registers to correct with 'C'.

Source: John Kennedy (918).

```
01 LBL "MIO"
02 STOP
03 LBL A
04 "START REG?"
05 AVIEW
06 STOP
07 STO 07
08 "DIM: R?↑C?"
09 AVIEW
10 STOP
11 STO 08
12 X<>Y
13 STO 09
14 SF 09
15 GTO 01
16 LBL B
17 CF 09
18 LBL 01
19 CF 29
20 RCL 07
21 STO 04
22 RCL 08
23 RCL 09
24 *
25 STO 03
26 LBL 02
27 RCL 04
28 XEQ "M4"
29 FIX 0
30 " ("
31 ARCL Y
32 "⊢,"
33 ARCL X
34 "⊢)="
35 FC? 09
36 GTO 03
37 "⊢?"
38 AVIEW
39 TONE 0
40 STOP
41 STO IND 04
42 GTO 04
43 LBL 03
44 FIX IND 05
45 ARCL IND 04
46 AVIEW
47 LBL 04
48 ISG 04
49 STO X
50 DSE 03
51 GTO 02
52 BEEP
53 RTN
54 LBL C
55 XEQ "M5"
56 " R"
57 ARCL X
58 AVIEW
59 STO 04
60 1
61 STO 03
62 CF 09
63 GTO 02
64 END
(133 bytes)
```

11-4 FINDING DETERMINANTS & INVERSES; SOLVING SYSTEMS OF EQUATIONS ("RRM"): This routine will transform a matrix into row reduced eschelon form; this means it will calculate determinants and inverses and will solve systems of equations. It will handle these three matrix problems, either individually or simultaneously, and uses the technique known as partial pivoting which helps reduce round off error. The only limitation on the size of the matrix is the number of available data registers. "RRM" can even be applied to more than one matrix in data memory. "RRM" calls on the following routines, which must also be in program memory: "M1" - "M5" (11-1), "BE" (10-14 or 11-1), "BX" (10-18) and "QR" (or 11-1). The Matrix Input/Output Routine (11-3) is useful for loading, reviewing and correcting matrices, and is used in the examples below. Before executing "RRM", the number of the starting register of the matrix (10 or greater) must be stored in R07, the number of columns must be stored in R08, and the number of rows must be stored in R09. "MIO" will have stored these values, if executed just prior to this routine.

FLAGS:
10: used

REGISTERS:
R01: determinant
R02: used
R03: used
R04: used
R07: starting register
R08: # of columns
R09: # of rows
ALPHA: used

Example 1: First work example 1 of "MIO"; when the matrix is loaded, XEQ "RRM"; execution time will be about 40 seconds; when it ends, get into "MIO", then press 'B' in USER Mode to display the final matrix, which is:

$$\left[\begin{array}{ccc:c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right]$$

The solution is x = 2, y = 3, z = -1. The determinant of the square coefficient is stored in R01: det. = 150.0000 .

Example 2: Work example 2 of "MIO". When the matrix is loaded, XEQ "RRM"; when execution ends, get into "MIO" and press 'B' in USER Mode to display the matrix:

$$\left[\begin{array}{ccc:ccc:c} 1 & 0 & 0 & 0.0631 & 0.0291 & 0.0388 & 0.5000 \\ 0 & 1 & 0 & 0.1068 & 0.1262 & -0.1650 & 2.0000 \\ 0 & 0 & 1 & 0.0162 & 0.1100 & 0.0356 & 0.3333 \end{array}\right]$$

The last column contains the solutions of the system of equations and would be interpreted as x = 1/2, y = 2, z = 1/3. The determinant of the coefficient of the matrix can be recalled from R01: det. = 618. The inverse of the original matrix is the 3x3 matrix in the middle. "DF" (14-1) can be used to convert the full value of these decimals (note they're in Registers 13-15, 20-22 & 27-29) to fractions:

$$\begin{bmatrix} 13/206 & 3/103 & 4/103 \\ 11/103 & 13/103 & -17/103 \\ 5/309 & 34/309 & 11/309 \end{bmatrix}$$

If only the determinant of a matrix is desired, a square matrix is all that "RRM" requires. If only the inverse of a matrix is desired, input as in Example 2 of "MIO" but leave out the final (right hand) column of constants. "RRM" is just as useful for systems of equations which do not have unique solutions. If the determinant in R01 is zero, then the system of equations may have no solutions or an infinite number of solutions. Since "RRM" returns the row reduced echelon form, the final matrix will always be row equivalent to the original. The final matrix may then be used to tell immediately where parameters should be inserted and any and all solutions may then be immediately determined. The coeffieient matrix need not be square for "RRM" to operate on it.

Source: John Kennedy (918).

```
01 LBL "RRM"   13 RCL 08    25 1 E3       37 1/X        49 XEQ "M1"   61 X=Y?
02 .           14 RCL 04    26 /          38 RCL M      50 RCL 01     62 GTO 07
03 STO 03      15 X>Y?      27 +          39 INT        51 CHS        63 RCL 02
04 STO 04      16 RTN       28 RCL 08     40 XEQ "M4"   52 STO 01     64 RCL 04
05 SIGN        17 RCL 09    29 1 E5       41 RDN        53 LBL 07     65 XEQ "M5"
06 STO 01      18 RCL 03    30 /          42 STO 02     54 ISG 02     66 RDN
07 SF 10       19 X>Y?      31 +          43 XEQ "M2"   55 STO X      67 RCL IND T
08 LBL 05      20 RTN       32 XEQ "BX"   44 RCL 02     56 RCL 09     68 CHS
09 ISG 03      21 RCL 04    33 RCL IND M  45 ST- 02     57 RCL 02     69 XEQ "M3"
10 LBL 06      22 XEQ "M5"  34 ST* 01     46 RCL 03     58 X>Y?       70 GTO 07
11 ISG 04      23 X<> Z     35 X=0?       47 X=Y?       59 GTO 05     71 END
12 STO X       24 XEQ "M5"  36 GTO 06     48 GTO 07     60 RCL 03       (124 bytes)
```

CHAPTER XII

SORTING

12-1 QUICKSORT ("QS"): This routine sorts data in R01 - Rnn. Have 'nn' in X before executing "QS". It clears Flag 21, sets FIX 3, uses Flag 04. It doesn't work as well for data already sorted. The block currently being sorted is displayed while the routine is running. "QS" uses a few registers above Register 'nn' for scratch; to sort 'N' numbers, up to $N+1+\log_2 N$ registers are needed. For example, to sort data in R01 - R32, key '32', XEQ "QS". When BEEP sounds and "DONE" appears, a review of the data registers will show a 'used' control number in R00, the data in R01-32 sorted in ascending order, 'garbage' in R33-36, and the contents of R37 and above unchanged. A sort of 32 numbers ordered randomly takes about 1 minute; a 'resort' of this sorted data will take about 2½ minutes. Source: Mike Hale (4457) (PPC CJ, V7N2 P39).

```
01 LBL "QS"
02 STO 00
03 CF 21
04 FIX 3
05 CF 04
06 RCL 00
07 1 E3
08 /
09 1
10 +
11 ST+ 00
12 STO IND 00
13 LBL 15
14 RCL IND 00
15 VIEW X
16 INT
17 LASTX
18 FRC
19 1 E3
20 *
21 STO Z
22 X<>Y
23 -
24 15
25 X>Y?
26 XEQ 99
27 FS?C 04
28 GTO 24
29 RCL IND Z
30 R↑
31 RCL L
32 X<>Y
33 LBL 16
34 R↑
35 CLX
36 RCL IND Z
37 R↑
38 X<=Y?
39 GTO 18
40 LBL 17
41 ISG T
42 STOP
43 RDN
44 RDN
45 X<=Y?
46 GTO 23
47 GTO 16
48 LBL 18
49 STO IND T
50 X<>Y
51 STO IND Z
52 GTO 20
53 LBL 19
54 R↑
55 RCL IND Y
56 X<=Y?
57 GTO 21
58 LBL 20
59 DSE Z
60 CLA
61 RDN
62 RDN
63 X<=Y?
64 GTO 23
65 GTO 19
66 LBL 21
67 STO IND T
68 X<>Y
69 STO IND Z
70 GTO 17
71 LBL 22
72 1 E3
73 /
74 +
75 X<> IND 00
76 ISG 00
77 STOP
78 STO IND 00
79 ENTER
80 RTN
81 LBL 23
82 1
83 -
84 RCL IND 00
85 INT
86 X<>Y
87 X>Y?
88 XEQ 22
89 RDN
90 RDN
91 1
92 +
93 RCL IND 00
94 FRC
95 1 E3
96 *
97 X>Y?
98 XEQ 22
99 LBL 24
100 DSE 00
101 GTO 15
102 "DONE"
103 AVIEW
104 BEEP
105 RTN
106 LBL "IST"
107 LBL 99
108 SF 04
109 ISG IND 00
110 LBL 25
111 RCL IND 00
112 RCL IND X
113 X<>Y
114 1
115 -
116 INT
117 LBL 26
118 X<>Y
119 RCL IND Y
120 X<=Y?
121 GTO 27
122 ISG Z
123 BEEP
124 STO IND Z
125 RDN
126 X<>Y
127 2
128 -
129 X≠0?
130 GTO 26
131 STO Z
132 LBL 27
133 CLX
134 1
135 ST+ Z
136 RDN
137 STO IND Y
138 ISG IND 00
139 GTO 25
140 END
```

(248 bytes)

12-2 STACK SORT ("S1"): This routine sorts data in the X, Y, Z & T Registers; it won't work if an Alpha string is in any of these registers. Before execution, have Flag 10 clear for a descending sort (largest value returned to X, smallest to T), or have Flag 10 set for an ascending sort. "S1" clears Flag 10. To eliminate this feature, change step 29 (FS?C 10) to FS? 10. Source: PPC ROM.

```
01 LBL "S1"
02 X>Y?
03 X<>Y
04 RDN
05 X>Y?
06 X<>Y
07 R↑
08 X>Y?
09 X<>Y
10 R↑
11 X<=Y?
12 GTO 01
13 X<>Y
14 RDN
15 X<=Y?
16 GTO 02
17 X<>Y
18 RDN
19 X<=Y?
20 GTO 03
21 X<>Y
22 LBL 03
23 RDN
24 RDN
25 GTO 01
26 LBL 02
27 R↑
28 LBL 01
29 FS?C 10
30 RTN
31 RDN
32 X<> Z
33 END
```

(46 bytes)

12-3 SYNTHETIC QUICKSORT ("S2" & "S3"): These routines work as quickly for presorted data as they do for randomly-ordered data. All registers above the block being sorted are left undisturbed. For both routines, have a 'bbb.eee' control number defining the block to be sorted in X before entering the routine; the control number is returned to X after execution. The stack and the Alpha Register are used, as well as Flag 10. "S2" (Small Array Sort) is faster for 32 or fewer registers; it uses no numeric registers, and so will sort any block of data registers. It is executed as a subroutine by "S3". "S3" (Large Array Sort) is faster for more than 32 registers; it uses Registers 01 & 02, so the block to be sorted must begin with Register 03 or a higher-numbered register. A comparison of some execution times for "S2" and "S3", respectively: for 5 registers: 5 sec. vs. 12 sec.; for 32 registers: 44 sec. each; for 96 registers: 4 min. 47 sec. vs. 2 min. 48 sec. Source: Ray Evans (4928) (PPC ROM).

```
01 LBL "S3"
02 STO O
03 SF 10
04 LBL 09
05 STO 02
06 LBL 05
07 STO 01
08 FRC
09  E3
10 *
11 STO N
12 RCL 01
13 INT
14 -
15  E
16 +
17 STO M
18 RCL 01
19  E-4
20 +
21 .
22 RCL X
23 LBL 14
24 RCL IND Z
25 +
26 ISG Y
27 ""
28 ISG Z
29 GTO 14
30 X<>Y
31 /
32 RCL N
33 X<>Y
34 RCL 01
35 STO T
36 LBL 13
37 X<> IND N
38 LBL 11
39 X<Y?
40 GTO 12
41 DSE N
42 ""
43 DSE M
44 GTO 13
45 GTO 06
46 LBL 12
47 X<> IND T
48 ISG T
49 ""
50 DSE M
51 GTO 11
52 LBL 06
53 X<> IND Z
54 FRC
55 R↑
56 INT
57 +
58 LBL 10
59 ENTER
60 FRC
61  E3
62 *
63 X<>Y
64 -
65 31
66 X>Y?
67 GTO 01
68 LASTX
69 FS?C 10
70 GTO 09
71 GTO 05
72 LBL 01
73 LASTX
74 XEQ "S2"
75 INT
76 RCL O
77 INT
78 X=Y?
79 GTO 08
80 RDN
81 RCL 01
82 INT
83 X<>Y
84 X>Y?
85 GTO 07
86 RCL 02
87 INT
88 X<>Y
89 X>Y?
90 GTO 07
91  E
92 -
93  E3
94 /
95 RCL O
96 INT
97 +
98 SF 10
99 GTO 09
100 LBL 07
101  E
102 -
103  E3
104 /
105 +
106 GTO 10
107 LBL 08
108 LASTX
109 BEEP
110 RTN
111 LBL "S2"
112 CF 10
113 STO M
114 INT
115  E99
116 STO P
117 ENTER
118 ENTER
119 ENTER
120 LBL 04
121 X<> M
122 STO N
123 X<> M
124 LBL 00
125 X<> IND N
126 X<Y?
127 GTO 01
128 ISG N
129 GTO 00
130 SF 10
131 GTO 02
132 LBL 01
133 R↑
134 X<Y?
135 GTO 01
136 X<> P
137 R↑
138 R↑
139 ISG N
140 GTO 00
141 GTO 02
142 LBL 01
143 X<> P
144 X<Y?
145 GTO 01
146 RDN
147 X<> P
148 RDN
149 ISG N
150 GTO 00
151 GTO 02
152 LBL 01
153 X<> T
154 X<Y?
155 X<>Y
156 R↑
157 X<> P
158 X<> T
159 ISG N
160 GTO 00
161 LBL 02
162 R↑
163 X<> IND M
164 X<> P
165 ISG M
166 X<> IND M
167 R↑
168 ISG M
169 X<> IND M
170 R↑
171 ISG M
172 X<> IND M
173 FS?C 10
174 GTO 03
175 R↑
176 ISG M
177 X<> IND M
178 LBL 03
179 ISG M
180 GTO 04
181 LASTX
182 END
```

(282 bytes)

12-4 FINDING SMALLEST (OR LARGEST) OF THREE OR MORE NUMBERS ("SORT"): This routine finds the smallest of three numbers; to find the largest of three numbers, substitute "X<Y?" for "X>Y?" (steps 04 & 07). To extend to four or more numbers, add sets of steps like 04-06 for each additional number. Source: Bill Kolb (265) (BP 67/97).

```
LBL "SORT", RCL 01, RCL 02, X>Y?, X<>Y, RCL 03, X>Y?, X<>Y, RTN          (16 bytes)
```

12-5 SYNTHETIC ALPHABETIZE X & Y ("AL"): This routine alphabetizes Alpha strings in the X and Y Registers (in ACCHR order). The contents of registers Z & T are lost. Indirect use: With the first register number in Y and the second in X, the contents of the registers pointed to by the numbers in X & Y will be switched in necessary to put them in alphabetical order (the string first in alphabetical order will be returned to the register pointed to by X). RDN twice to recover the values that were in X & Y. Source: Wickes (3735), Jarett (4360) & Cheeseman (4381) (PPC ROM).

```
01 LBL "AL"
02 CLA
03 CF 10
04 XEQ 14
05 XEQ 14
06 X=Y?
07 XEQ 13
08 CLA
09 SF 10
10 X<=Y?
11 CF 10
12 FC?C 25
13 GTO 12
14 X<=Y?
15 RTN
16 X<> IND T
17 X<> IND Z
18 X<> IND T
19 RTN
20 LBL 12
21 X<=Y?
22 GTO 12
23 R↑
24 X<> Z
25 LBL 12
26 R↑
27 R↑
28 RTN
29 LBL 13
30 R↑
31 R↑
32 SF 10
33 XEQ 14
34 LBL 14
35 SF 25
36 ARCL IND Y
37 FC? 25
38 ARCL Y
39 "⊦◆◆◆◆◆"
40 ASTO L
41 ARCL L
42 "⊦◆◆◆◆◆"
43 .
44 FC? 10
45 GTO 14
46 STO N
47 "⊦◆◆"
48 LBL 14
49 STO M
50 "⊦◆◆"
51 X<> N
52 END
(107 bytes)
```

CHAPTER XIII

RANDOM NUMBERS

13-1 (PI + SEED)^3 RANDOM NUMBER GENERATOR ("RAN"): Store a fractional seed in R10 before executing "RAN". Example: key '.05101975, STO 10, XEQ "RAN"'; in FIX 4 Mode, '0.5416' is returned to X; to repeat, press R/S; '0.9649' results. Source: Jim Butterfield (1076) (65 NOTES, V4N8P4).

LBL "RAN", RCL 10, PI, +, ENTER, X↑2, *, FRC, STO 10, RTN. (16 bytes)

13-2 SHORTEST, FASTEST RANDOM NUMBER GENERATOR: Use 'LBL any, RCL any, R-D, FRC, STO any, RTN'. If desired, another "R-D" could be added. The previous seed must be stored in the selected register; it can be any integer or decimal, except 0 or pi (or its multiples). It generates numbers between 0 and 1 uniformly. Source: Valentin Albillo (4747) (PPC CJ, V7N6P35).

13-3 RANDOM NUMBER GENERATOR ("RN"): "RN" is a random number generator and can be used to generate uniformly-distributed pseudorandom numbers in the range 0<r<1. Input required is a register pointer in X; this is the number of the register which holds the seeds. This register should be initialized with a random decimal between 0 and 1 before the first time this routine is called. The output leaves the new seed in X as well as in the data register. Source: Don Malm (1362) (65 NOTES, V4N8P4).

		Inputs:	Outputs:
01 LBL "RN"	06 +		
02 RCL IND X	07 FRC	T: T	T: Y
03 9821	08 STO IND Y	Z: Z	Z: Y
04 *	09 RTN	Y: Y	Y: Reg. Pointer
05 .211327	(25 bytes)	X: Reg. Pointer	X: Next Seed

13-4 GAUSSIAN RANDOM NUMBER GENERATOR ("GN"): This routine yields a Gaussian (bell-shaped) distribution where the mean and the standard deviation are specified by the user. "GN" calls "RN" (13-3), and hence requires a register pointer in X when called. An initial seed must also be stored in the register pointed to by the number in X. "GN" leaves two Gaussian random numbers in the stack: one in X, one in Y. This routine must be used in Degree Mode. Source: Kiyoshi Akima (3456) & John Kennedy (918) (PPC CJ, V7N8P11; PPC ROM).

		Inputs:	Outputs:
01 LBL "GN"	10 *		
02 XEQ "RN"	11 R↑	T: T	T: Reg. Pointer
03 LN	12 RCL 07	Z: Z	Z: Reg. Pointer
04 ST+ X	13 *	Y: Y	Y: Random No. #2
05 CHS	14 P-R	X: Reg. Pointer	X: Random No. #1
06 SQRT	15 RCL 06	L: L	L: Mean
07 X<>Y	16 ST+ Z	R06: Mean	R07: Standard Deviation
08 XEQ "RN"	17 +		
09 360	18 RTN		(33 bytes)

To eliminate the restriction of being in Degree Mode, replace line 09 (360) with '1, ASIN, 4, *'. This change will cost only one additional byte. (In any trig mode, the value calculated will represent a full circle, and the P-R function will receive its data in the correct format.) Source: Larry Trammell (6824).

13-5 USING INDIRECT ADDRESSING TO TEST A RANDOM NUMBER GENERATOR ("TR" & "RNG"):

The "ISG IND X" function can be used to provide data for testing a random number generating routine that returns a decimal fraction to X. The value returned is multiplied by 10 to put the first decimal digit in the integer portion of the number, then "ISG IND X" increments the register (R00-09) corresponding to that digit. These registers must be cleared first. An example is "TR" below. It clears R00-12, then prompts for the number of times the random number generator, labeled "RNG", is to be executed. It also prompts for a fractional seed, then stores it in R10. The number of times "RNG" has been executed is displayed as execution continues. When execution is complete, BEEP sounds and the calculator turns OFF. To see the results, turn the calculator ON and view R00-09. The number in R00 is the number of times a zero was generated, the number in R01 is the number of times a one was generated, and so on. A sample random number generator, "RNG", is also listed. The results for testing "RNG", first for 100 trials, then for 1000 trials, are shown below right; in both cases, a seed of .2579846319 was used. Source: Bill Kolb (265) & John Dearing (2791). See PPC CJ, V7N4P16.

```
01 LBL "TR"
02 CF 21
03 FIX 0
04 12
05 STO 00
06 CLX
07 LBL 14
08 STO IND 00
09 DSE 00
10 GTO 14
11 "NO. TRIALS ?"
12 PROMPT
13 STO 11
14 "FRAC. SEED ?"
15 PROMPT
16 STO 10
17 LBL 13
18 XEQ "RNG"
19 10
20 *
21 ISG IND X
22 BEEP
23 1
24 ST+ 12
25 VIEW 12
26 DSE 11
27 GTO 13
28 BEEP
29 PSE
30 OFF
40 END
   (78 bytes)
```

```
01 LBL "RNG"
02 RCL 10
03 997
04 *
05 FRC
06 STO 10
07 END
   (17 bytes)
```

REG	TRIALS 100	1000
R00	13	103
R01	6	94
R02	14	118
R03	13	99
R04	8	105
R05	9	92
R06	9	88
R07	8	107
R08	9	104
R09	11	90

R10 = seed
R11 = loop counter
R12 = no.-of-executions-completed counter

CHAPTER XIV

FRACTIONS & ROUNDING

14-1 DECIMAL TO FRACTION ("DF"): The input accepted by this routine is any decimal value in X (it may include an integer portion); the output is a fraction, Y/X, whose approximation to the decimal input will agree to at least 'n' places, where n is stored in R07 before execution. The numerator will be in Y and the denominator in X. Source: John Kennedy (918) (PPC CJ, V7N8P11; PPC ROM).

```
01 LBL "DF"
02 STO 08
03 INT
04 0
05 STO 09
06 1
07 STO 10
08 RCL 08
09 R↑
10 X=Y?
11 GTO 08
12 ST- Y
13 LBL 07
14 RDN
15 1/X
16 ENTER
17 INT
18 -
19 RCL 09
20 RCL 10
21 STO 09
22 LASTX
23 *
24 +
25 STO 10
26 RCL 08
27 *
28 FIX 0
29 RND
30 STO Z
31 RCL 10
32 /
33 RCL 08
34 -
35 FIX IND 07
36 RND
37 X≠0?
38 GTO 07
39 RCL Z
40 LBL 08
41 RCL 10
42 SIGN
43 ST* 10
44 *
45 RCL 10
46 END
```

(61 bytes)

14-2 DECIMAL-TO-FRACTION DRIVER ("DFD"): This routine 'drives' the decimal to fraction routine ("DF") above (14-1). It is fully printer compatible: it operates the same with no printer, with a printer plugged in and on, and with a printer plugged in but off. When "DFD" is executed, it prompts for 'n' and for the decimal, executes "DF", and then formats and displays the results. Press R/S to see the actual value of the fraction; press backarrow to return to a FIX 2 display, or press R/S again to rerun the routine. If 'n' or the decimal are to be the same as before, skip the prompt for the unchanged value with R/S. Example: for n = 5 and decimal = pi (3.141592654), the fraction = 355/133 = 3.141592920 . Source: John Dearing (2791).

```
01 LBL "DFD"
02 RCL 07
03 "N ?"
04 PROMPT
05 STO 07
06 RCL 08
07 "DECIMAL?"
08 PROMPT
09 XEQ "DF"
10 FIX 0
11 CF 29
12 CLA
13 ARCL Y
14 "⊢/"
15 ARCL X
16 FIX 9
17 SF 29
18 PROMPT
19 /
20 CLA
21 ARCL X
22 FIX 2
23 PROMPT
24 GTO "DFD"
25 END
```

(61 bytes)

14-3 REDUCE FRACTIONS ("RED"): To use, key in numerator, ENTER, denominator, XEQ "RED". The routine pauses to show the greatest common divisor (delete steps 16-19 to eliminate this feature), then stops to display the fraction in reduced form. Source: John Dearing (2791) (PPC CJ, V7N4P32).

```
01 LBL "RED"
02 STO 01
03 X<>Y
04 STO 00
05 X<>Y
06 LBL 01
07 STO Z
08 MOD
09 X≠0?
10 GTO 01
11 +
12 ST/ 00
13 ST/ 01
14 FIX 0
15 CF 29
16 "GCD="
17 ARCL X
18 AVIEW
19 PSE
20 CLA
21 ARCL 00
22 "⊢/"
23 ARCL 01
24 AVIEW
25 FIX 2
26 SF 29
27 CLX
28 END
```

(53 bytes)

14-4 ROUND TO NEAREST FRACTION ("NF"): Have the number to be rounded in Y and the decimal fraction in X, then execute "NF". If the number to be rounded is negative, the decimal fraction must also be negative. Examples: to round '24.56' to the nearest whole number, key 24.56, ENTER, 1, XEQ "NF"; see '25.00'. To round '2.29' to the nearest half, key 2.29, ENTER, .5, XEQ "NF"; see '2.50'. To round '-12.79' to the nearest third, key 12.79, CHS, ENTER, 3, 1/X, CHS, XEQ "NF"; see '-12.67'.

LBL "NF", /, LASTX, X<>Y, .5, +, INT, *, RTN. (15 bytes)

14-5 NEAREST FRACTION ROUND: Multiply by reciprocal of fraction, add .5 if positive or subtract .5 if negative, take the integer, then divide by reciprocal of the fraction. Examples:

Pos. # round to nearest integer: ..., .5, +, INT, ...
Pos. # round to nearest half: ..., 2, *, .5, +, INT, 2, /, ...
Neg. # round to nearest third: ..., 3, *, .5, -, INT, 3, /, ...
Pos. or Neg. # round to nearest fourth: ..., CF 14, X<0?, SF 14, 4, *, .5, FC? 14, +, FS?C 14, -, INT, 4, /, ...

14-6 STEP-FUNCTION ROUND:

Pos. # round up to the next integer if there's a fractional part (FIX 2 Mode) (so 13.00 → 13, but 13.01 → 14): ..., .99, +, INT, ... For FIX 3 Mode, use '.999'; for FIX 4, use '.9999', and so on.
Another pos. # round up to the next integer if there's a fractional part (any display mode): use ..., ENTER, FRC, X≠0?, SIGN, +, INT, ... Rounds negative numbers down if there's a fractional part.

14-7 FRACTIONAL ARITHMETIC ("F+", "F-", "F*", "F/", "RE" & "MX"): This program is a set of routines for addition, subtraction, multiplication, division and reduction of fractions, plus the conversion of an improper fraction to a mixed number. SF 10 before execution to display/print fraction. Synthetic steps 96, 99, 101 & 103 can be replaced with similar functions using any convenient numeric register, such as R00. If the Improper Fraction to Mixed Number routine ("MX") isn't wanted, steps 55-104 may be eliminated.

"F+", F-", "F*", "F/": Addition, subtraction, multiplication and division of fractions: input is the fractions in T,Z & Y,X; output is the fraction in Y,X (& the fraction [Y/X] in the display if Flag 10 is set). Example: solve 2/5 ÷ -3/4. Solution: key 2, ENTER, 5, ENTER, 3, CHS, ENTER, 4, XEQ "F/": output is 15 in X and -8 in Y (and "-8/15" in the display if Flag 10 is set).

"RE": Reduce fraction. Input is the fraction in Y,X; output is the reduced fraction in Y,X (and the fraction [Y/X] in the display if Flag 10 is set). Example: reduce 45/925. Solution: key 45, ENTER, 925, XEQ "RE"; output is 185 in X and 9 in Y (and "9/185" in the display if Flag 10 is set).

"MX": Change an improper fraction (numerator >= denominator) to a mixed number (integer + fraction). Works for positive or negative fractions. Input is the improper fraction in Y,X; output is the integer in X and the fraction in Z,Y (and the mixed number [X Z/Y] in the display if Flag 10 is set). Example: change -747/126 to a mixed number. Solution: key 747, CHS, ENTER, 126, XEQ "MX"; output is -5 in X, 14 in Y, and 13 in Z (and "-5 13/14" in the display if Flag 10 is set).

"GD": Greatest Common Divisor. See routine 15-13.
"QR": Quotient & Remainder. See routine 15-12.
"VA": View Alpha. See routine 4-1.

Suggested key assignments (A-E, H-J):

A	"F+"	"F-"	"F*"	"F/"	"RE"	E
F			"QR"	"GD"	"MX"	J

Source: John Kennedy (918) (Adapted from PPC CJ, V7N8P8-11 & PPC ROM).

```
01 LBL "F-"
02 CHS
03 LBL "F+"
04 ST* T
05 X<> Z
06 *
07 ST+ Z
08 X<> L
09 *
10 GTO 12
11 LBL "F/"
12 X<>Y
13 LBL "F*"
14 ST* Z
15 X<> T
16 *
17 X<>Y
```

[continued]

```
18 LBL "RE"
19 LBL 12
20 RCL Y
21 RCL Y
22 XEQ 13
23 ST/ Z
24 /
25 FC? 10
26 RTN
27 FIX 0
28 CF 29
29 CLA
30 ARCL Y
31 "⊦/"
32 ARCL X
33 FIX 2
34 SF 29
35 XEQ "VA"
36 RTN
37 LBL "GD"
38 LBL 13
39 MOD
40 LASTX
41 X<>Y
42 X≠0?
43 GTO 13
44 +
45 RTN
46 LBL "VA"
47 SF 25
48 PRA
49 SF 25
50 FS?C 21
51 CF 25
52 AVIEW
53 FC?C 25
54 SF 21
55 RTN
56 LBL "MX"
57 CF 11
58 FS?C 10
59 SF 11
60 XEQ 12
61 X<>Y
62 CF 14
63 X<0?
64 SF 14
65 ABS
66 X<>Y
67 XEQ "QR"
68 X<>Y
69 FS?C 14
70 CHS
71 LASTX
72 X<>Y
73 FS?C 11
74 SF 10
75 FC? 10
76 RTN
77 FIX 0
78 CF 29
79 CLA
80 ARCL X
81 X<> Z
82 X=0?
83 GTO 14
84 "⊦ "
85 ARCL X
86 "⊦/"
87 ARCL Y
88 LBL 14
89 X<> Z
90 FIX 2
91 SF 29
92 XEQ "VA"
93 RTN
94 LBL "QR"
95 X<>Y
96 STO O
97 X<>Y
98 MOD
99 ST- O
100 LASTX
101 ST/ O
102 CLX
103 X<> O
104 X<>Y
105 END
```

(219 bytes)

14-8 DECIMAL TO 'RULER FRACTION' CONVERSION ("F-D" & "D-F"): This routine converts between decimal fractions and 'ruler' fractions--fractions having denominators that are integral powers of 2, like the markings on English (inches) rulers (1/8, 3/32, etc). No numeric data registers or flags are used. This routine is useful when you need to do arithmetic on a fraction: just convert to a decimal, perform the desired arithmetic operations, then convert back to a fraction. To convert a fraction to a decimal: key in the integer portion (even if zero), ENTER, numerator, ENTER, denominator; XEQ "F-D". See the decimal in X. To convert a decimal to a fraction: key in decimal, XEQ "D-F". See fraction in display. In X,Y,Z,T order, the stack will contain the denominator, numerator, integer, and original decimal.

As written, "D-F" will approximate the fraction (decimal) to the nearest 32nd. Change line 10 (32) to '16' for the nearest 16th, etc. Assignment suggestion: ASN "F-D" to -73 (SCI) & "D-F" to -74 (ENG). Then think of pressing SCI to convert to a 'scientific' (decimal) form, or pressing ENG to convert to an 'English' (fractional) form. Example: to the nearest 32nd of an inch, what is 60% of 5½ inches? Solution: key 5, ENTER, 1, ENTER, 2, XEQ "F-D" (see 5.50); key 60, % (see 3.30); XEQ "D-F", see the answer, "3 5/16". Source: Richard Kimmel (6003) (PPC CJ, V7N10P24).

```
01 LBL "F-D"
02 /
03 +
04 RTN
05 LBL "D-F"
06 ENTER
07 FRC
08 FIX 0
09 CF 29
10 32
11 ENTER
12 X<> Z
13 *
14 RND
15 X=0?
16 GTO 02
17 X=Y?
18 GTO 04
19 ENTER
20 LBL 01
21 CLX
22 2
23 ST/ Z
24 /
25 ENTER
26 FRC
27 X=0?
28 GTO 01
29 CLX
30 2
31 ST* Z
32 *
33 LBL 02
34 CLA
35 R↑
36 INT
37 X=0?
38 X=Y?
39 ARCL X
40 "⊦ "
41 RDN
42 X=0?
43 GTO 03
44 ARCL X
45 "⊦/"
46 ARCL Y
47 LBL 03
48 X<> Z
49 RDN
50 FIX 2
51 SF 29
52 PROMPT
53 RTN
54 LBL 04
55 CLX
56 1
57 ST+ T
58 CLX
59 GTO 02
60 END
```

(96 bytes)

14-9 EVEN ROUND ("ER"): The mainframe function "RND" always rounds up when the digit to be rounded is exactly '5' (followed by zeros). This routine will leave the last digit retained even, rounding up or down as appropriate. See the example at right, showing the results of using "ER" and "RND" on various numbers in FIX 2 Mode.

#	ER	RND
.545	.54	.55
.555	.56	.56
-.545	-.54	-.55
-.555	-.56	-.56

LBL "ER", 2, /, RND, 2, *, RTN (12 bytes)

CHAPTER XV

ARITHMETIC & ALGEBRA

15-1 <u>SUM OF INTEGERS ("ΣI")</u>: This routine finds x where $x = 1+2+3+\ldots+n$, by solving the equation $x = [n(n+1)]/2 = [n^2+n]/2$. Example: the rule of 78's used in financial interest problems comes about by considering 12 months per year: $78 = 1+2+\ldots+12$. Source: Richard Nelson (1) (65 NOTES, V1N4P3).

```
LBL "ΣI", X↑2, LASTX, +, 2, /, RTN                                  (12 bytes)
```

15-2 <u>SUM OF SQUARES ("ΣS")</u>: This routine finds x, where $x = 1^2 + 2^2 + \ldots + n^2$ (the sum of the squares of integers from 1 to n) without looping, by solving the equation $x = [n(n+1)(2n+1)]/6$. Example: for n = 10, x = 385. Source: Dom Tocci (189) (65 NOTES, V2N8P3).

```
LBL "ΣS", X↑2, LASTX, +, LASTX, 2, *, 1, +, *, 6, /, RTN            (18 bytes)
```

Another version using register arithmetic with the L Register: Source: Bill Kolb (265):

```
LBL "ΣS", X↑2, ST+ X, LASTX, +, ST* L, LASTX, +, 6, /, RTN          (18 bytes)
```

15-3 <u>SUM OF CUBES ("Σ3")</u>: This routine finds x, where $x = 1^3 + 2^3 + \ldots + n^3$ by solving the equation $x = \{[n(n+1)]/2\}^2$. Example: for n = 5, x = 225. Source: Bill Kolb (265) (BP 67/97).

```
LBL "Σ3", X↑2, LASTX, +, 2, /, X↑2, RTN                             (13 bytes)
```

15-4 <u>SUM OF THE DIGITS OF AN INTEGER ("ΣD")</u>: This routine finds the sum of the digits of the integer in the X Register. Example: the sum of the digits of the integer 1234556789 (note the 2 fives) is 50. Source: John Kennedy (918) (PPC J, V5N7 P4).

```
01 LBL "ΣD"     04 LBL 01     07 ST+ 01     10 X≠0?      13 ST* 01
02 STO 01       05 10         08 INT        11 GTO 01    14 RCL 01
03 ST- 01       06 /          09 ST- 01     12 10        15 RTN        (27 bytes)
```

15-5 <u>CONVERT A REAL NUMBER TO A DECIMAL OR AN INTEGER ("-DEC" & "-INT")</u>: To convert the number in X to a decimal with the same digits, XEQ "-DEC"; to convert the number in X to an integer with the same digits, XEQ "-INT". Example: Key '123.45', XEQ "-DEC", see 0.12345 in FIX 5 Mode; XEQ "-INT", see 12345.00000. Source: James Davidson (547) (65 NOTES, V3N1P2)

```
01 LBL "-DEC"    05 X=Y?    09 GTO 01        13 LASTX    17 *
02 LBL 01        06 RTN     10 LBL "-INT"    14 X=Y?     18 GTO 02
03 FRC           07 10      11 LBL 02        15 RTN      19 RTN
04 LASTX         08 /       12 INT           16 10                     (39 bytes)
```

15-6 <u>REVERSE INTEGER ("IV")</u>: This routine reverses the order of the digits of the integer in the X Register. Source: James Davidson (547) (65 NOTES, V2N10P10).

```
01 LBL "IV"     04 LBL 01     07 LASTX     10 ST* 01     13 GTO 01
02 STO 01       05 FRC        08 INT       11 /          14 RCL 01
03 ST- 01       06 ST+ 01     09 10        12 X≠0?       15 RTN        (25 bytes)
```

15-7 FIBONACHI SERIES ("FB"): When "FB" is executed, the calculator will display the Fibonachi Series, in which each number is the sum of the previous two numbers. The series starts: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... Source: James Davidson (547) (65 NOTES, V3N9P14).

LBL "FB", FIX 0, 1, ENTER, 0, LBL 01, PSE, X<>Y, +, LASTX, GTO 01, RTN (19 bytes)

15-8 TO INTRODUCE A SMALL ERROR INTO X: Method 1: EEX, 8, CHS, +or-. Method 2: SQRT, X↑2. Method 2 doesn't work for many numbers, including perfect squares. Source: Bill Kolb (265) (BP 67/97). See 22-24.

15-9 CHAIN ARITHMETIC: Source: HP KEY NOTES, V4N3P11.

Chain Subtraction: To repeatedly subtract a constant value, 'k', from the base number, 'n', in X, key 'k', CHS, ENTER, ENTER, ENTER, 'n'; then press '+' the desired number of times. Example: key 5, CHS, ENTER, ENTER, ENTER, 1000; press + repeatedly and see 995, 990, etc.

Chain Addition: Delete the CHS instruction above. Example: to repeatedly add 5 to 1000, key 5, ENTER, ENTER, ENTER, 1000; press + repeatedly to see 1005, 1010, etc.

Chain Division: To repeatedly divide a constant, 'k', into a base number, 'n', in X, key 'k', 1/X, ENTER, ENTER, ENTER, 'n'; then press * the desired number of times. Example: key 5, 1/X, ENTER, ENTER, ENTER, 1000; press * repeatedly and see 200, 40, 8, etc.

Chain Multiplication: Delete the '1/X' instruction above. Example: key 5, ENTER, ENTER, ENTER, 1000; press '*' repeatedly and see 5000, 25000, 125000, etc.

15-10 REPEATED MULTIPLICATION OR DIVISION BY 10: To repeatedly multiply the value in X by 10, use EEX, 3, X<>Y, %, %, %, ... Each repeated '%' multiplies the value by 10. Similarly, to repeatedly divide the value in X by 10, use EEX, 1, X<>Y, %, %, %, ... If the value to be repeatedly multiplied (or divided) is in a numeric data register, replace the 'X<>Y' instruction with 'RCL nn', where 'nn' is the number of the data register. Source: Curt Rostenback (382) (PPC J, V5N3P15).

15-11 INTEGER DIVIDE ("I/"): This routine returns the integer quotient of y/x to X and the remainder of dividing y by x to Y. Z is preserved and X is in LASTX. Source: David Motto (2339) (PPC CJ, V7N7P14). Positive values of x & y only.

LBL "I/", RCL Y, X<>Y, MOD, ST- Y, X<>Y, LASTX, /, RTN (16 bytes)

15-12 QUOTIENT & REMAINDER ("QR"): This routine replaces Y with the integer quotient of y/x, and replaces X with the remainder of dividing y by x. Y & Z are preserved; X is in LASTX. Part of the Alpha Register, the O Register, is used as a scratch register and cleared afterward. This will not affect any text already in Alpha as long as it contains no more than 14 characters. For a nonsynthetic version, replace the steps using the O Register (steps 03, 06, 08 & 10) with the same operations using any numeric register, such as R00. Source: Roger Hill (4940) (PPC ROM).

01 LBL "QR"	03 STO O	05 MOD	07 LASTX	09 CLX	11 X<>Y	
02 X<>Y	04 X<>Y	06 ST- O	08 ST/ O	10 X<> O	12 RTN	(21 bytes)

15-13 GREATEST COMMON DIVISOR ("GCD" & "GD"): These routines will compute the greatest common divisor of the values in X and Y. Source: John Kennedy (918) (PPC J, V6N5P31; PPC CJ, V7N8P8). "GCD" preserves T; "GD" preserves Z & T.

LBL "GCD", LBL 01, STO Z, MOD, X≠0?, GTO 01, +, RTN (16 bytes)

LBL "GD", LBL 01, MOD, LASTX, X<>Y, X≠0?, GTO 01, +, RTN (15 bytes)

15-14 PRIME DIVISOR, TEST IF PRIME, GENERATE PRIMES, PRIME FACTORS ("PD", "TP", "GT" & "PF"): "PD" (Prime Divisor) gives the next prime divisor of an integer greater than or equal to a specified trial divisor which may be 2 or any odd number. Input an integer 'n' greater than or equal to 2 in Y and a possible trial divisor 'd' in X, where 'd' is any prime number including 2, or any odd number greater than 1. The output from the routine is 'n' in Y and either 'd' or the next odd integer larger than 'd', whichever divides evenly into 'n', in X. Pressing R/S after execution of the routine will give the next prime factor of the integer. "TP" (Test if Prime) tests if the integer in X is prime; it returns the number to X so you can immediately execute "PF" if "NOT PRIME" appears. "GT" (Generate a Table of Primes) generates a list of prime numbers beginning with 1. "PF" (Prime Factors) gives the prime factors of an integer. Source: John Kennedy (918) (PPC CJ, V7N3P6; V7N9P11).

```
01 LBL "PD"
02 LBL 01
03 RCL Y
04 RCL Y
05 /
06 X<Y?
07 GTO 02
08 FRC
09 X=0?
10 GTO 03
11 CLX
12 2
13 X=Y?
14 DSE X
15 ABS
16 +
17 GTO 01
18 LBL 02
19 RDN
20 LBL 03
21 RDN
22 RTN
23 ST/ Y
24 GTO 01
25 LBL "TP"
26 VIEW X
27 2
28 XEQ 01
29 "  NOT PRIME"
30 X=Y?
31 ASHF
32 AVIEW
33 X<>Y
34 RTN
35 LBL "GT"
36 1
37 VIEW X
38 PSE
39 2
40 VIEW X
41 PSE
42 1
43 LBL 04
44 2
45 ST+ Y
46 XEQ 01
47 X=Y?
48 VIEWX
49 RDN
50 GTO 04
51 LBL "PF"
52 2
53 LBL 05
54 XEQ 01
55 VIEW X
56 PSE
57 X=Y?
58 RTN
59 ST/ Y
60 GTO 05
61 END
                (117 bytes)
```

15-15 NEXT PRIME ("NP"): This routine gives the next prime divisor of an integer greater than or equal to a specified trial divisor which may be 2 or any odd number. Pressing R/S automatically gives the next prime divisor. Key integer, ENTER, trial divisor; XEQ "NP". "NP" is valid for 10-digit positive integers. The divisor the routine returns will be prime provided 'n' has no prime factors strictly smaller than 'd'. If "NP" is executed from the keyboard, when the next divisor is returned, immediately pressing R/S will cause "NP" to continue searching for the next factor. This may be repeated, but when the routine returns '1', there are no more factors of 'n'. All computations are carried out in the stack; no numeric data registers are used. Tests 3.5 divisors/second. To use "NP" from the keyboard to see if an integer is prime, use '2' as the starting trial divisor; if the original number is returned, then that number is prime. To use "NP" from the keyboard to find all the prime factors of an integer, key in the integer, ENTER, 2, XEQ "NP"; repeatedly press R/S to see the prime factors, until '1' is returned.

Example: the number 40,013,933 is known to have only two prime factors, one of which is greater than 5000; find them. Solution: we may start with any odd number greater than 5000, so key 40,013,933, ENTER, 5001, XEQ "NP". After about 45 seconds, execution stops with '5309' in the display. Pressing R/S twice more gives '7537' and '1'. Hence 40,013,933 = 5309 * 7537. Source: Phi Trinh (6171) (PPC ROM).

```
01 LBL "NP"
02 LBL 11
03 RCL Y
04 SQRT
05 LASTX
06 X<> Z
07 LBL 09
08 X>Y?
09 R↑
10 R↑
11 X<>Y
12 MOD
13 X=0?
14 GTO 10
15 X<> L
16 2
17 X=Y?
18 SIGN
19 +
20 GTO 09
21 LBL 10
22 R↑
23 LASTX
24 X>Y?
25 ENTER
26 RTN
27 ST/ Y
28 GTO 11
29 END
                (43 bytes)
```

15-16 REPLACE X WITH ITS EXPONENT OR MANTISSA ("XPN" & "MAN"): To replace the value in X with its exponent, XEQ "XPN"; the range for X is 10^{-99} - $9.999999884 \times 10^{99}$. To replace the value in X with its mantissa (X must be positive), XEQ "MAN". Source: Rob Jung (2455) (PPC J, V5N7P6) & John Martellaro (1896) (PPC J, V5N8P9).

```
01 LBL "XPN"
02 LOG
03 ENTER
04 ENTER
05 FRC
06 -
07 X=Y?
08 RTN
09 X>0?
10 RTN
11 X=0?
12 RTN
13 1
14 -
15 RTN
16 LBL "MAN"
17 ENTER
```

[continued]

18 LOG	20 10↑X	22 GTO 01	24 /	26 /	
19 INT	21 X<=Y?	23 10	25 LBL 01	27 END	(43 bytes)

15-17 VIEW MANTISSA ("VMAN"): XEQ "VMAN" to see all the digits of the mantissa of the number in X, in SCI 9 or FIX 9 Modes. If assigned to the shifted ENTER key (-41), it will match the location of the similar function on the HP-34C. Uses the Alpha Register; press backarrow (correction) key to restore the X-Register display. Source: HP KEY NOTES, V4N3P12.

LBL "VMAN", CLA, ARCL X, AVIEW, RTN (13 bytes)

15-18 SYNTHETIC MANTISSA & EXPONENT ("MANT" & "EXP"): "MANT" replaces X by its mantissa, and saves Y, Z, T & L. "EXP" replaces X by its exponent, and saves Y, Z & T. Only the Alpha Register is used. "The trick used in the routine is to replace the last byte of the number by Hex 50 (the letter P), and then to divide or multiply by E50 depending on the sign of the exponent." Source: Roger Hill (4940) (PPC CJ, V7 N8P2).

01 LBL "MANT"	05 ASTO M	09 E50	13 ST/ M	17 XEQ "MANT"	21 END
02 CLA	06 "├P"	10 SF 25	14 X<> M	18 ST/ N	
03 CF 24	07 STO N	11 ST* M	15 RTN	19 X<> N	
04 STO M	08 CLX	12 FC?C 25	16 LBL "EXP"	20 LOG	(56 bytes)

15-19 SYNTHETIC VIEW MANTISSA, MANTISSA & EXPONENT ("VM", "EX" & "MT"): "VM" views the full mantissa of the number in X; the stack is undisturbed. "EX" and "MT" replace the number in X with its exponent or mantissa, respectively, leaving the rest of the stack undisturbed. Line 37 ("├") is nonstandard; it is decimal 242, 127, 160. Source: PPC ROM. "VM" clears Flag 21. Roger Hill (4940) & Dave Kaplan (   ).

01 LBL "VM"	08 FIX 9	15 CLA	22 LASTX	29 LBL "MT"	36 X=0?
02 CF 21	09 VIEW M	16 X≠0?	23 X<> M	30 CLA	*37 "├"
03 XEQ "MT"	10 STO d	17 "θΩ"	24 ASHF	31 STO M	38 CLX
04 X<> M	11 RDN	18 INT	25 "├◆Δ"	32 ASTO M	39 ST+ M
05 RDN	12 LASTX	19 X≠0?	26 ST- M	33 INT	40 X<> M
06 VIEW O	13 RTN	20 CLA	27 X<> M	34 X≠0?	41 END
07 RCL d	14 LBL "EX"	21 RDN	28 RTN	35 "├◆"	(84 bytes)

15-20 RAPID RATIO SOLUTIONS ("R1", "R2" & "R3"): These routines solve the ratio equation A/B = C/D for any term. "R1" is a stack solution; "R2" is a register solution; "R3" works either way (press A in USER Mode for a register solution, or press B for a stack solution). Enter or store each term in the order A, B, C & D (storing in Registers 01, 02, 03 & 04 respectively); enter the unknown term as zero. Source: Chris Stevens (3005) (PPC J, V5N7P4).

01 LBL "R1"	10 /	01 LBL "R2"	10 X≠0?	01 LBL "R3"	10 RDN
02 X<>Y	11 /	02 RCL 01	11 RDN	02 LBL A	11 X≠0?
03 X≠0?	12 END	03 RCL 02	12 RDN	03 RCL 01	12 RDN
04 RDN		04 RCL 04	13 /	04 RCL 02	13 X≠0?
05 X≠0?		05 RCL 03	14 /	05 RCL 03	14 RDN
06 RDN		06 X≠0?	15 END	06 RCL 04	15 RDN
07 X≠0?		07 RDN		07 LBL B	16 /
08 RDN		08 X≠0?		08 X<>Y	17 /
09 RDN	(19 bytes)	09 RDN	(22 bytes)	09 X≠0?	18 END (27 bytes)

15-21 CUBE ROOT OF ANY NUMBER, POSITIVE OR NEGATIVE ("CURT"): This method saves bytes over a version that uses a flag test to decide whether to change the sign of the final result. Change the 3 to any other odd integer to get any nth root (n odd). Source: Valentin Albillo (4747).

LBL "CURT", SIGN, LASTX, ABS, 3, 1/X, Y↑X, *, RTN (16 bytes)

15-22 SQUARE ROOT OF THE SUMS OF VARIOUS SQUARES: Source: Bill Kolb (265) (BP67/97).

To find $\sqrt{X^2 + Y^2}$, use R-P (where X is the value in the X Register, and Y is the value in the Y Register).

To find $\sqrt{Y^2 + 2X^2}$, use R-P, LASTX, R-P. For $\sqrt{X^2 + 2Y^2}$, X<>Y first.

To find $\sqrt{A^2 + B^2 + C^2 + D^2 + \ldots}$, use RCL A, RCL B, R-P, RCL C, R-P, RCL D, R-P, ...

15-23 FAST FACTORIAL FACTOR FINDER ("FFFF"): Key 'n' (an integer from 3 to 9999), XEQ "FFFF" to find the factors of n!. See the first two factors; if no printer, R/S for each succeeding pair until BEEP sounds. Example: 12! = 2↑10 • 3↑5 • 5↑2 • 7↑1 • 11↑1. Source: Joel Lichtenwalner (2957) (PPC J, V5N8P46).

```
01 LBL "FFFF"     15 XEQ 02   29 SQRT     43 X=0?     57 INT        71 SF 00
02 CF 00          16 LBL 01   30 LBL 00   44 RTN      58 ST+ 03     72 RTN
03 SF 21          17 RCL 02   31 ENTER    45 -        59 X≠0?       73 LBL 05
04 FIX 0          18 RCL 01   32 FRC      46 1        60 GTO 04     74 FS?C 00
05 CF 29          19 2        33 X=0?     47 +        61 FC? 00     75 AVIEW
06 ADV            20 +        34 RTN      48 /        62 CLA        76 SF 29
07 "FACTORS OF "  21 X>Y?     35 -        49 GTO 00   63 FS? 00     77 FIX 2
08 ARCL X         22 GTO 05   36 1        50 LBL 03   64 "├ , "     78 RDN
09 "├ FACT.:"     23 XEQ 02   37 X=Y?     51 CLX      65 ARCL 01    79 CLD
10 AVIEW          24 GTO 01   38 GTO 03   52 STO 03   66 "├↑"       80 BEEP
11 STO 02         25 LBL 02   39 *        53 RCL 02   67 ARCL 03    81 END
12 2              26 STO 01   40 /        54 LBL 04   68 FS? 00
13 XEQ 02         27 ENTER    41 ENTER    55 RCL 01   69 AVIEW
14 3              28 ENTER    42 FRC      56 /        70 FC?C 00      (142 bytes)
```

15-24 TO MULTIPLY TWO COMPLEX NUMBERS IN THE STACK ("MC"): This routine accepts two complex numbers ($z_1 = a_1 + b_1 i$ and $z_2 = a_2 + b_2 i$) stored in the stack as follows: X: a_1 ; Y: b_1 ; Z: a_2 ; T: b_2 . After the routine is executed, the result is: X: real part, Y: imaginary part. This routine uses no numeric data registers; it does not use trigonometric functions; and it does not use P-R or R-P, so is much faster than routines that do. Executes in 0.4 seconds. Source: Valentin Albillo (4747).

```
01 LBL "MC"   04 ST* Y   07 R↑      10 X<> L   13 RDN   16 END
02 STO L      05 X<> Z   08 ST* Y   11 R↑      14 +
03 R↑         06 ST* Z   09 ST* L   12 -       15 R↑        (30 bytes)
```

15-25 QUADRATIC EQUATION, REAL ROOTS, STACK SOLUTION ("QEQ"): This routine finds the real roots of a quadratic equation of the form $ax^2 + bx + c = 0$, by solving the equation $x = \{-b \pm \sqrt{b^2 - 4ac}\} / 2a$. Complex roots give "DATA ERROR". No numeric data registers or flags are used. To use, key a, ENTER, b, ENTER, c, XEQ "QEQ"; the roots will be returned to the X & Y Registers (X<>Y to see the second root). Source: Robert Groom (5127).

```
01 LBL "QEQ"   04 ST+ X   07 ENTER   10 R↑     13 ST- Z
02 X<> Z       05 /       08 ENTER   11 -      14 +
03 ST/ Z       06 CHS     09 X↑2     12 SQRT   15 END      (26 bytes)
```

15-26 COMPLEX QUADRATIC EQUATION PLUS DISPLAY ("QE" & "PRQE"): "QE" finds the roots of a quadratic equation, whether real or complex (of the form u ± iv). To use, key a, ENTER, b, ENTER, c, XEQ "QE". If there are two real roots, they will be returned to X & Y (and Flag 04 will be cleared); if the roots are complex, the real part, 'u', will be returned to X, and the imaginary part, 'iv', will be returned to Y (and Flag 04 will be set [its annunciator will be on]). "QE" is usable as a subroutine. SIZE 000. Source: Robert Groom (5127).

After executing "QE", R/S to execute "PRQE"; this routine formats and displays/prints the results of "QE". It sets Flag 21 and clears Flag 04. Source: adapted from a routine by John Herzfeld (5428).

01 LBL "QE"	08 ENTER	15 ABS	22 LBL "PRQE"	29 "ROOT 2= "	36 "⊢ +- "
02 CF 04	09 ENTER	16 SQRT	23 SF 21	30 ARCL Y	37 ARCL Y
03 X<> Z	10 X↑2	17 ST- Z	24 FS?C 04	31 AVIEW	38 "⊢ I"
04 ST/ Z	11 R↑	18 X<>Y	25 GTO 14	32 RTN	39 AVIEW
05 ST+ X	12 -	19 FC? 04	26 "ROOT 1= "	33 LBL 14	40 END
06 /	13 X<0?	20 +	27 ARCL X	34 CLA	
07 CHS	14 SF 04	21 RTN	28 AVIEW	35 ARCL X	(92 bytes)

For automatic display/print without pressing R/S, delete steps 21 & 22.

15-27 BIG FACTORIALS ("BF"): This routine approximates n! for large values of n by using the first three terms of a Stirling series approximation. The number of significant digits in the calculated result is at least 10 minus the number of digits in the power of ten. No numeric data registers are used and there is no looping, so execution is fast. The input is 'n' in the X Register; the output is the mantissa in X and the power of 10 ("decapower") in Y. Source: Larry Trammell (6824).

01 LBL "BF"	07 *	13 LN	19 ENTER	25 -	31 LASTX
02 ENTER	08 LASTX	14 +	20 R↑	26 +	32 FRC
03 LN	09 ST+ X	15 R↑	21 X↑2	27 10	33 10↑X
04 1	10 PI	16 12	22 /	28 LN	34 END
05 -	11 *	17 *	23 30	29 /	
06 RCL Y	12 SQRT	18 1/X	24 /	30 INT	(46 bytes)

15-28 Y↑X FOR LARGE VALUES OF X & Y ("BYX"): Enter X and Y normally (key Y, ENTER, X), then XEQ "BYX". The mantissa of the result is returned to X; the power of ten ("decapower") is returned to Y. Accuracy is limited by the use of logarithms. Example: find 25 to the 75th power. Solution: key 25, ENTER, 75, XEQ "BYX"; see 7.006493122 (in FIX 9 Mode) in X; X<>Y to see 104.0000000. Hence, 25^{75} = 7.006493122 x 10^{104}. Source: Bill Derrick (1393) (PPC J, V5N7P4).

LBL "BYX", X<>Y, LOG, *, INT, LASTX, FRC, 10↑X, RTN (15 bytes)

15-29 POLYNOMIALS ("POLY"): This routine solves the equation $y = ax^3 \pm bx^2 \pm cx \pm d$. Put a '+' or a '-' in the routine for each '±' above, as appropriate. To expand the polynomial, add series of steps like steps 08-10 (RCL nn, ±, *). Source: Bill Kolb (265) (BP 67/97).

					R00: x
01 LBL "POLY"	05 ENTER	09 + or -	13 *		R01: a
02 RCL 00	06 RCL 01	10 *	14 RCL 04		R02: b
03 ENTER	07 *	11 RCL 03	15 + or -		R03: c
04 ENTER	08 RCL 02	12 + or -	16 END	(25 bytes)	R04: d

15-30 POLYNOMIAL EVALUATION ("PE"): This routine evaluates a polynomial of arbitrary order. ($y = \ldots + ax^3 + bx^2 + cx + d$). To use, key control number, ENTER, argument (x), XEQ "PE". The control number (bbb.eee) defines the block of registers containing the coefficients (a,b, etc); the coefficient of the highest order term is in R'bbb', and the constant coefficient is in R'eee'. The value of the polynomial (y) is returned to X, and the value of the argument is returned to Y.

Example: for $y = 3x^4 - 2x^3 - 5x^2 + 6x + 12$, find y if x = 7. Solution: store the coefficients in any block, say R01-05: 3, STO 01, 2, CHS, STO 02, 5, CHS, STO 03, 6, STO 04, 12, STO 05. Next, key 1.005, ENTER, 7, XEQ "PE"; see '6,326.00'. Source: Larry Trammell (6824).

01 LBL "PE"	03 CLX	05 R↑	07 RCL IND Y	09 ISG Y	11 R↑	13 END
02 STO Z	04 LBL 01	06 *	08 +	10 GTO 01	12 X<>Y	(24 bytes)

15-31 POLYNOMIAL MULTIPLY ("P*"): This routine will return to a block beginning with a specified register the coefficients of the resulting polynomial when two given polynomials are multiplied. The control or index number defining this output block is returned to X and also to R03. The resulting polynomial can be evaluated for a specific argument using the routine "PE" (15-30). The coefficients of each of the polynomials to be multiplied must first be stored as a block of data registers, with the coefficient of the highest-numbered term in the lowest-numbered (bbb) register of the block. Registers 01-03 must also be loaded as follows before execution:

R01: Index to the coefficients of the 1st polynomial (bbb.eee).
R02: Index to the coefficients of the 2nd polynomial (BBB.EEE).
R03: Pointer to the 1st register of the output block.

This routine changes the value in R01; it uses R00 and Flag 10. Registers 04 and above are available for the two input- and one output-blocks. The number of registers needed for output is one more than the degree of the resulting polynomial.

Example: find $(3x^3 + 2x^2 - 5)(4x^2 + 6)$. Solution: for the 1st polynomial, the coeffients are: $a_1 = 3$, $b_1 = 2$, $c_1 = 0$, $d_1 = -5$; for the 2nd, $a_2 = 4$, $b_2 = 0$, $c_2 = 6$. Four registers are needed to store the coefficients of the 1st polynomial (use R04-R07), and three for the 2nd (use R08-R10). Load data registers as shown at right. The first register of the output block is to be R11 (so '11' is stored in R03). The degree of the resulting polynomial is 5 ($3x^3 \cdot 4x^2 = 12x^5$), so 6 registers are needed for the output (R11-R16); hence a minimum SIZE 017 is needed.

R01: 4.007
R02: 8.010
R03: 11
R04: 3
R05: 2
R06: 0
R07: -5
R08: 4
R09: 0
R10: 6

Now execute "P*". When execution stops, see '11.016' in X (the index to the output block). Review R11-16 (use "PRREGX" with a printer) to see

R11= 12, R12= 8, R13= 18, R14= -8, R15= 0, R16= -30.

Therefore the resulting polynomial is

$$12x^5 + 8x^4 + 18x^3 - 8x^2 - 30.$$

If "PE" is in memory, you can evaluate this expression for a specific value of x: for x = 2, 'RCL 03, 2, XEQ "PE"'; see '594'; for x = 3, 'RCL 03, 3, XEQ "PE"; see '3948'.

Source: Larry Trammell (6824).

```
01 LBL "P*"      09 RCL 03        17 DSE X     25 CLX           33 STO IND Y     41 CF 10
02 RCL 03        10 RCL IND 01    18 1 E3      26 STO IND Y     34 FC? 10        42 END
03 INT           11 XEQ 12        19 /         27 GTO 01        35 ST+ IND Y
04 STO 00        12 ISG 03        20 RCL 00    28 LBL 12        36 X<> L
05 STO 03        13 LBL 00        21 +         29 RCL IND Z     37 ISG Y
06 SF 10         14 ISG 01        22 STO 03    30 X<>Y          38 LBL 00
07 LBL 01        15 GTO 02        23 RTN       31 *             39 ISG Z
08 RCL 02        16 RDN           24 LBL 02    32 FS? 10        40 GTO 12        (71 bytes)
```

15-32 DECIBEL ADDITION & SUBTRACTION ("dB+" & "dB-"): Uses no data registers or flags. To use, key in sound pressure levels in decibels (dB_1, ENTER, dB_2); then, to add, XEQ "dB+", or to subtract, XEQ "dB-". Source: HP-41C Users' Library Solutions Heating, Ventilating & Air Conditioning, pp 65-68.

```
01 LBL "dB+"     06 XEQ 00     11 10         16 AVIEW      21 /          26 END
02 XEQ 00        07 -          12 *          17 RTN        22 X<>Y
03 +             08 ABS        13 CLA        18 LBL 00     23 10↑X
04 GTO 01        09 LBL 01     14 ARCL X     19 10         24 X<>Y
05 LBL "dB-"     10 LOG        15 "├ dB"     20 ST/ Z      25 10↑X       (53 bytes)
```

CHAPTER XVI

GEOMETRY, TRIG & CALCULUS

16-1 TO KEEP A VECTOR POSITIVE: To keep a vector, 'r', positive, use 'RCL θ, RCL r, P-R, R-P'. Source: Dave Wilder (452) (BP 67/97).

16-2 TAN +90 DEGREES: Prevents overflow at multiples of +90 degrees: 'SQRT, X↑2, TAN'. Source: Dave Wilder (452). Use 'RAD, D-R, TAN, DEG' for ±90 degrees.

16-3 KEEPING ANGLES LESS THAN 90° OR 180°: To keep an angle less than 90°, use 'SIN, ASIN'; to keep an angle less than 180°, use 'COS, ACOS'. Source: Dave Wilder (452) (BP 67/97).

16-4 SUPPLEMENT OF AN ANGLE, KEPT WITHIN ±180° ("SUP"): Supplement $\theta = 180° - \theta$. With θ in X, XEQ "SUP" to convert it to its supplement. Source: Bill Kolb (265) (BP 67/97).

LBL "SUP", -1, P-R, R-P, X<>Y, CHS, RTN (14 bytes)

16-5 ELIMINATING DISPLAY OF 60 MIN OR SEC: Use 'RND, HR, HMS'. Source: John Martellaro (1896) (65 NOTES, ~~V7~~V4N9P1). Assums FIX 4. Display ONLY. Losses

---

16-6 COS & SIN OF X SIMULTANEOUSLY: Use '1, P-R'. These two steps put cos x in X and sin x in Y. Source: Joachim Bolz (401) (65 NOTES, V2N9P25).

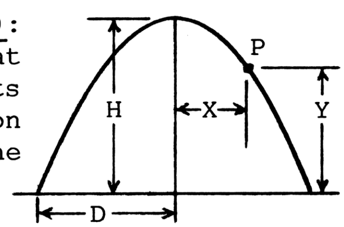
16-7 ARCTAN Y/X: Instead of '/, ATAN', use 'R-P, RDN'. This avoids division by zero and distinguishes between -y/x and y/-x. Source: Dave Wilder (452) (BP 67/97).

16-8 BOUNDING ANGLES: Routine A keeps angles between 0° and 360°; routine B keeps angles between plus and minus 180°. Source: HP KEY NOTES, V3N4P9.

$0° <= \theta < 360°$: LBL A, 360, P-R, R-P, X<>Y, X<0?, +, RTN (11 bytes)

$-180° < \theta <= 180°$: LBL B, 1, P-R, R-P, X<>Y, RTN (7 bytes)

16-9 ELEVATION OF A POINT ON A PARABOLA, STACK SOLUTION ("PN"): To find the elevation 'Y' of a point 'P' on a parabola, at distance 'X' from the center axis, XEQ "PN"; the routine prompts for D, H & X, then finds the elevation 'Y'. For another point on the same parabola, key the new value of 'X', then press R/S; the new elevation will be found. May repeat. Source: John Dearing (2791). Equation: $Y = H[1-(X/D)^2]$.

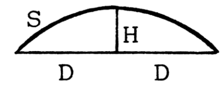



```
01 LBL "PN"    05 PROMPT    09 LBL 14    13 RDN     17 *         21 R↑        25 GTO 14
02 "D?"        06 1         10 R↑        14 X↑2     18 "Y= "     22 LASTX     26 END
03 PROMPT      07 "X?"      11 /         15 -       19 ARCL X    23 1
04 "H?"        08 PROMPT    12 LASTX     16 X<>Y    20 PROMPT    24 R↑       (44 bytes)
```

16-10 LAW OF COSINES: $c = \sqrt{a^2 + b^2 - 2ab \cdot \cos\theta}$.

Use: RCL θ, RCL a, P-R, RCL b, -, R-P. Source: Bill Kolb (265).

16-11 AREA & LENGTH OF A RIGHT PARABOLIC SEGMENT ("AP" & "SP"): Input 'D', ENTER, 'H', XEQ "AP" for area or "SP" for length.




```
01 LBL "AP"   07 RTN        13 X<>Y    19 ENTER    25 /        31 /
02 *          08 LBL "SP"   14 X↑2     20 ENTER    26 LN       32 *
03 4          09 STO 01     15 4       21 RCL 01   27 RCL 02   33 +
04 *          10 X<>Y       16 *       22 ST+ X    28 X↑2      34 END
05 3          11 STO 02     17 +       23 +        29 RCL 01
06 /          12 X↑2        18 SQRT    24 RCL 02   30 ST+ X           (48 bytes)
```

16-12 AREA OF A REGULAR POLYGON: 'N' is the number of sides; 'S' is the length of a side; and 'R' is the radius of the circumscribed circle (center-to-vertex). "N↑S": Input N, ENTER, S; XEQ "N↑S". "N↑R": Input N, ENTER, R; XEQ "N↑R". Calculator must be in DEG Mode. Source: Hugh Kenner (103) (PPC CJ, V7N5P7).

```
01 LBL "N↑S"   06 LASTX   11 /          16 LBL "N↑R"   21 LASTX   26 /
02 X↑2         07 /       12 "A="       17 X↑2         22 /       27 "A="
03 X<>Y        08 TAN     13 ARCL X     18 X<>Y        23 SIN     28 ARCL X
04 *           09 /       14 AVIEW      19 *           24 *       29 AVIEW
05 180         10 4       15 RTN        20 360         25 2       30 END   (54 bytes)
```

16-13 AREA OF A REGULAR POLYGON, ANY TRIG MODE: Key in 'N' (number of sides), ENTER, 'R' (radius of circumscribed circle), XEQ "AR". Works in any trigonometric mode. Equation: $K = \frac{1}{2}NR^{2}$ SIN(360°/N). The sequence '1, ASIN, 4, *' replaces the 360° because it works in RAD and GRAD Modes as well.

```
01 LBL "AR"   04 4       07 /      10 X↑2   13 2
02 1          05 *       08 SIN    11 *     14 /
03 ASIN       06 RCL Z   09 X<>Y   12 *     15 END            (23 bytes)
```

16-14 SPHERICAL/RECTANGULAR COORDINATE CONVERSION; EULER TRANSFORMATIONS ("R-S", "S-R", & "ET"): Here is a pair of routines for transforming a coordinate triplet between spherical and rectangular coordinates. "R-S" is initialized by z, ENTER, y, ENTER, x; it returns r, θ, φ in Registers X-Y-Z. "S-R" requires φ, ENTER, θ, ENTER, r; it returns the rectangular coordinates to X-Y-Z. Both routines leave the T Register undisturbed.

```
01 LBL "R-S"   03 R↑      05 R-P   07 LBL "S-R"   09 X<> T   11 P-R
02 R-P         04 X<> T   06 RTN   08 P-R         10 RDN     12 END   (28 bytes)
```

The result of an arbitrary set of reference-axis rotations on the coordinates of a point in 2-dimensional space can be effected simply by adding a single angle α to the φ-coordinate of the point. In three dimensions, three such angles, often called "Euler Angles", are required to describe the result of an arbitrary set of rotations. Euler angle transformations by one convention or another are used, for example, in programs for perspective plotting, for predicting the position of astronomical objects in the night sky, and for orienting crystal lattices for diffraction studies. Hewlett-Packard PPC command architecture is particularly suited to this manually formidable calculation, as shown by the 41C/V routine below. Here, Euler angles α, β, and λ (by convention a z-axis, a y'-axis and a z"-axis rotation, respectively) are stored in R01, R02 & R03. Spherical coordinates of a point φ, ENTER, θ, ENTER, r are then changed by "ET" into the transformed r, θ, φ, making use of stack registers only. Since the value of coordinate r doesn't change under rotation, an arbitrary non-zero constant can be used instead of r. Source: Phil Fraundorf (1025) (PPC CJ, V7N8P26).

```
01 LBL "ET"   05 -      09 R-P      13 X<>Y   17 R↑       21 RDN
02 P-R        06 X<>Y   10 X<>Y     14 P-R    18 R-P      22 END
03 X<> Z      07 P-R    11 RCL 02   15 RDN    19 RCL 03
04 RCL 01     08 R↑     12 -        16 R-P    20 ST- T           (32 bytes)
```

16-15 <u>SOLVING INTEGER-SIDED RIGHT TRIANGLES ("IT")</u>: This routine will successively solve for all the integer-sided right triangles having a given integer as one side. If Flag 00 was cleared before execution, Tone 0 sounds and execution stops here ('One-Integer Mode'). If Flag 00 was set before execution, the integer is incremented and execution continues ('Continuous Mode'). If the printer is off or is not plugged in, R/S after the sides of a triangle are displayed. If printer is on, press R/S to stop execution. The display will show an asterisk after the number when it is the hypotenuse of the triangle. Since oblique triangles may be viewed as two right triangles having a common altitude, the routine is also useful in solving integer-sided oblique triangle problems.

Instructions: 1. CF 00 for 'One-Integer Mode', or SF 00 for 'Continuous Mode'. 2. XEQ "IT". 3. Key in integer, R/S. All integer-sided right triangles with the given integer as one side will be displayed/printed; if no printer, R/S after each. 4. In 'One-Integer Mode', Tone 0 sounds and 0.00 is displayed; R/S for a new case. In 'Continuous Mode', the integer is incremented by 1 and execution continues. To terminate Continuous Mode, press R/S if the routine is running, then CF 00 and XEQ 14 from the keyboard. Example: Key '5', XEQ "IT", see "5 12 13", "5* 3 4".

Source: Richard Smith (4856) (PPC CJ, V7N4P30).

```
01 LBL "IT"
02 SF 21
03 FIX 0
04 CF 29
05 "NO.?"
06 PROMPT
07 1
08 -
09 STO 06
10 GTO 00
11 LBL 14
12 FS? 00
13 GTO 00
14 CLX
15 CLD
16 CF 01
17 SF 29
18 FIX 2
19 TONE 0
20 RTN
21 GTO "IT"
22 LBL 00
23 CF 01
24 2
25 STO 05
26 ADV
27 1
28 ST+ 06
29 RCL 06
30 ENTER
31 STO 01
32 2
33 /
34 FRC
35 .5
36 X=Y?
37 GTO 12
38 RDN
39 X=0?
40 GTO 11
41 LBL 13
42 "NONE"
43 AVIEW
44 GTO 14
45 LBL 12
46 3
47 RCL 01
48 X<Y?
49 GTO 13
50 2
51 *
52 SF 01
53 GTO 10
54 LBL 11
55 4
56 RCL 01
57 X<Y?
58 GTO 13
59 LBL 10
60 X↑2
61 STO 00
62 FS? 01
63 GTO 08
64 LBL 09
65 RCL 00
66 RCL 05
67 RCL 01
68 X=Y?
69 GTO 05
70 RDN
71 /
72 STO 04
73 2
74 /
75 1
76 X=Y?
77 GTO 24
78 RDN
79 FRC
80 X=0?
81 XEQ 07
82 2
83 ST+ 05
84 GTO 09
85 LBL 08
86 RCL 00
87 RCL 05
88 /
89 STO 04
90 RCL 05
91 X<>Y
92 X<=Y?
93 GTO 05
94 RCL 04
95 FRC
96 X=0?
97 XEQ 07
98 4
99 ST+ 05
100 GTO 08
101 LBL 07
102 RCL 04
103 RCL 05
104 -
105 2
106 /
107 STO 02
108 RCL 05
109 +
110 STO 03
111 FS? 01
112 XEQ 06
113 CLA
114 ARCL 01
115 "├ "
116 ARCL 02
117 "├ "
118 ARCL 03
119 AVIEW
120 RTN
121 LBL 06
122 2
123 ST/ 02
124 ST/ 03
125 RTN
126 LBL 05
127 RCL 01
128 2
129 *
130 STO 00
131 0
132 STO 03
133 LBL 04
134 1
135 ST+ 03
136 RCL 00
137 RCL 03
138 -
139 LASTX
140 *
141 SQRT
142 STO 01
143 FRC
144 X=0?
145 GTO 03
146 RCL 00
147 2
148 /
149 RCL 03
150 X<Y?
151 GTO 04
152 GTO 14
153 LBL 03
154 2
155 ST/ 00
156 RCL 00
157 RCL 03
158 -
159 STO 02
160 RCL 01
161 X>Y?
162 GTO 14
163 CLA
164 ARCL 00
165 "├* "
166 ARCL 01
167 "├ "
168 ARCL 02
169 AVIEW
170 2
171 ST* 00
172 GTO 04
173 END
    (253 bytes)
```

16-16 <u>FUNCTIONS OF X AND $\sqrt{1 \pm X^2}$</u> : The range of x is $-1 < x < 1$.

For:	$\sqrt{1-x^2}$	$x / \sqrt{1-x^2}$	$\sqrt{1-x^2} / x$	$1 / \sqrt{1+x^2}$	$x / \sqrt{1+x^2}$
Use:	ACOS, SIN	ASIN, TAN	ACOS, TAN	ATAN, COS	ATAN, SIN

Source: Bill Kolb (265) (BP 67/97).

16-17 HYPERBOLIC FUNCTIONS ("SINH", "COSH", "TANH", "ASINH", "ACOSH" & "ATANH"):
Key in argument, execute appropriate function. For example, to compute the inverse hyperbolic tangent of x, XEQ "ATANH". No data registers are used; no local labels are used and there are no internal subroutines. The value in Y is returned to Y in each case; with "SINH" & "COSH", Z is returned to Z. Source: John Kennedy (918) (PPC CJ, V7N8P11).

```
01 LBL "SINH"
02 E↑X
03 ENTER
04 1/X
05 -
06 2
07 /
08 RTN
09 LBL "COSH"
10 E↑X
11 ENTER
12 1/X
13 +
14 2
15 /
16 RTN
17 LBL "TANH"
18 E↑X
19 ENTER
20 ENTER
21 1/X
22 ST- Z
23 +
24 /
25 RTN
26 LBL "ASINH"
27 ENTER
28 X↑2
29 1
30 +
31 SQRT
32 +
33 LN
34 RTN
35 LBL "ACOSH"
36 ENTER
37 X↑2
38 1
39 -
40 SQRT
41 +
42 LN
43 RTN
44 LBL "ATANH"
45 1
46 X<>Y
47 +
48 1
49 LASTX
50 -
51 /
52 SQRT
53 LN
54 END
   (66 bytes)
```

16-18 SOLVE ("SV"): This routine approximates a solution to an equation of the form f(x) = 0, using a Newton's (secant) method. Have the function name in R06 and the initial guess in X; then XEQ "SV". The output in X is the x-value which most closely makes f(x) = 0. Set Flag 10 to display successive approximations. Uses Registers 06-09. Source: Kennedy (918), Schwartz (2289) & Dennes (1757) (PPC ROM). Set display mode to desired accuracy.

```
01 LBL "SV"
02 STO 07
03 1
04 %
05 STO 09
06 CLST
07 LBL 04
08 RCL Z
09 STO 08
10 RCL 07
11 FS? 10
12 VIEW X
13 XEQ IND 06
14 ST* 09
15 ST- 08
16 RCL 09
17 RCL 08
18 X≠0?
19 /
20 STO 09
21 X<> 07
22 ST+ 07
23 RND
24 RCL 07
25 RND
26 X≠Y?
27 GTO 04
28 RCL 07
29 END
   (45 bytes)
```

16-19 INTEGRATE ("IG"): This routine duplicates the HP-34C Integrate function. Have the function name in R10, the lower limit of integration in Y and the upper limit in X. Accuracy depends on the display setting. Very slow! SF 10 to display successive approximations. Uses Registers 10-18. Source: Read Predmore (5184) (PPC ROM).

```
01 LBL "IG"
02 STO 17
03 X<>Y
04 -
05 4
06 /
07 STO 16
08 ST- 17
09 ST- 17
10 0
11 STO 15
12 STO 11
13 STO 18
14 SF 09
15 LBL 01
16 1
17 ENTER
18 2
19 STO 14
20 RCL 11
21 CHS
22 Y↑X
23 ST* 14
24 1
25 -
26 LBL 02
27 STO 12
28 X↑2
29 -
30 STO 13
31 2
32 +
33 RCL 12
34 *
35 RCL 16
36 *
37 RCL 17
38 +
39 XEQ IND 10
40 RCL 13
41 *
42 ST+ 15
43 1
44 RCL 12
45 RCL 14
46 +
47 X<Y?
48 GTO 02
49 RCL 11
50 STO 13
51 18
52 STO 12
53 1
54 ST+ 11
55 RCL 15
56 RCL 16
57 1.5
58 *
59 *
60 RCL 14
61 *
62 LBL 03
63 R↑
64 4
65 *
66 ENTER
67 DSE Y
68 X<> Z
69 ENTER
70 X<>IND 12
71 ST- Y
72 RND
73 X<> Z
74 /
75 RCL IND 12
76 +
77 ISG 12
78 STOP
79 DSE 13
80 GTO 03
81 STO IND 12
82 FS? 10
83 VIEW X
84 FS?C 09
85 GTO 01
86 RND
87 X≠Y?
88 GTO 01
89 LASTX
90 END
   (129 bytes)
```

16-20 FIRST DERIVATIVE ("FD"): This routine approximates the first derivative of a function at a point. Have the function loaded in memory as a program, with a global label of 6 or fewer characters. Have the function name in R11, the step size in R13 (0.01 is typical), and the x-value in X; then XEQ "FD". Source: Richard Schwartz (2289) (PPC CJ, V7N9P11-13, V7N10P10). [continued]

```
01 LBL "FD"
02 STO 12
03 RCL 13
04 +
05 XEQ IND 11
06 ST+ X
07 STO 14
08 RCL 12
09 RCL 13
10 ST+ X
11 +
12 XEQ IND 11
13 ST- 14
14 9
15 ST* 14
16 RCL 12
17 XEQ IND 11
18 11
19 *
20 ST- 14
21 RCL 12
22 RCL 13
23 3
24 *
25 +
26 XEQ IND 11
27 ST+ X
28 RCL 14
29 +
30 RCL 13
31 6
32 *
33 /
34 END
```

(52 bytes)

CHAPTER XVII

BASE CONVERSIONS

17-1 SYNTHETIC BASE CONVERSION ("BD" & "TB"): For positive integers, and $1 < b < 20$, these routines convert between numbers in base b and base 10. Store b in R06 before executing either routine. Source: George Eldridge (5575) (PPC CJ, V7N9P12; PPC ROM).

"BD" (Base b to base 10): Have the base b number in Alpha Register when routine is called.

Inputs:	Outputs:
Alpha: n_b	X: n_{10}
R06: b	Alpha: (cleared)

```
01 LBL "BD"
02 CLST
03 LBL 01
04 "├ "
05 X<> O
06 X=0?
07 GTO 01
08 X<> M
09 R↑
10 X<> N
11 "├◆Δ"
12 X<> N
13 RDN
14 X<> M
15  E
16 *
17 39
18 -
19 X>0?
20 DSE X
21 9
22 +
23 X<0?
24 GTO 02
25 X<>Y
26 RCL 06
27 *
28 +
29 .
30 GTO 01
31 LBL 02
32 RDN
33 CLA
34 RTN
```

"TB" (Base 10 to base b): Have the base 10 number in X Register when routine is called.

Inputs:	Outputs:
X: n_{10}	Alpha: n_b
R06: b	X: 0

```
35 LBL "TB"
36 "'             "
37 RCL M
38 X<>Y
39 LBL 03
40 ENTER
41 INT
42 RCL 06
43 MOD
44 9
45 -
46 X>0?
47 ISG X
48 LBL 04
49 39
50 +
51 10↑X
52 STO O
53 "├      "
54 CLX
55 X<> O
56 X<> N
57 STO M
58 RDN
59 RCL 06
60 /
61 INT
62 X≠0?
63 GTO 03
64 LBL 05
65 "├ "
66 CLX
67 RCL M
68 X≠Y?
69 GTO 05
70 CLX
71 X<> O
72 X<> N
73 STO M
74 CLST
75 AVIEW
76 END
```

(139 bytes)

NOTE: Line 11 is decimal 243, 127, 0, 8; line 36 is decimal 254, 39, and then thirteen 32's; line 53 is append 6 spaces.

17-2 STACK USED TO CONVERT TO BASE 10: To convert a positive integer in any base to base 10, load the stack with the number of the original base. It is not required that the original base be integral or even positive. Working from left to right, key the first digit of the integer and follow with "*" (multiply). Key in the next digit and follow with "+, *" (add, multiply). Continue keying in the digits, following each with "+, *" until the rightmost digit is keyed; follow it with "+" only. The resulting number in base 10 is now in the display. Example: convert 72305 in base 8 to base 10: key 8, ENTER, ENTER, ENTER; 7, *; 2, +, *; 3, +, *; 0, +, *; 5, +: see '29893' in the display. Source: Paul Fields (3114) (PPC J, V6N7P23).

17-3 OCTAL-DECIMAL CONVERSIONS FOR REAL NUMBERS: There may be some error in the result if the number is irrational in octal, or if the precision of the calculator is exceeded during calculation. The stack is destroyed, but no numeric data registers are used. Source: HP KEY NOTES, V4N1P7.

```
01 LBL "ROCT"
02 ENTER
03 INT
04 OCT
05 RCL Y
06 FRC
07 1073741824
08 *
09 INT
10 OCT
11 1 E10
12 /
13 +
14 RTN
15 LBL "RDEC"
16 ENTER
17 INT
18 DEC
19 RCL Y
20 FRC
21 1 E10
22 *
23 INT
24 DEC
25 1073741824
26 /
27 +
28 END
```

(70 bytes)

17-4 FAST DECIMAL-HEX ("DX"): Limited to integers in base 10 in the range 0 - 65,535. Meant to be used to compute addresses in a computer with up to 64K of memory. To use: 1. XEQ "DX". 2. Input 'D' (integer in base 10), R/S. 3. For a new case, go to step 2. Source: John Kennedy (918) (PPC CJ, V7N4P22).

```
01 LBL "DX"
02 "0"
03 ASTO 00
04 "1"
05 ASTO 01
06 "2"
07 ASTO 02
08 "3"
09 ASTO 03
10 "4"
11 ASTO 04
12 "5"
13 ASTO 05
14 "6"
15 ASTO 06
16 "7"
17 ASTO 07
18 "8"
19 ASTO 08
20 "9"
21 ASTO 09
22 "A"
23 ASTO 10
24 "B"
25 ASTO 11
26 "C"
27 ASTO 12
28 "D"
29 ASTO 13
30 "E"
31 ASTO 14
32 "F"
33 ASTO 15
34 LBL 00
35 "INPUT D, R/S"
36 PROMPT
37 LBL 01
38 65535
39 X<>Y
40 X>Y?
41 GTO 00
42 FIX 0
43 " "
44 ARCL X
45 "⊢ DEC"
46 AVIEW
47 4096
48 /
49 CLA
50 ARCL IND X
51 FRC
52 16
53 *
54 ARCL IND X
55 FRC
56 16
57 *
58 ARCL IND X
59 FRC
60 16
61 *
62 RND
63 ARCL IND X
64 "⊢ HEX"
65 AVIEW
66 STOP
67 GTO 01
68 END
```

(148 bytes)

17-5 SYNTHETIC HEX TO DECIMAL ("XD"): This routine works only for a 2-digit input in Alpha, and returns the decimal equivalent to X. For converting hex byte numbers to decimal in the Byte Table. No checking for invalid input. Source: Roger Hill (4940) (PPC ROM).

```
01 LBL "XD"
02 "⊢◆Δ"
03 RCL M
04  E2
05 XEQ "QR"
06 29
07 ST- Z
08 -
09 .9
10 ST* Z
11 *
12 INT
13 X<>Y
14 INT
15 16
16 *
17 +
18 RTN
19 LBL "QR"
20 X<>Y
21 STO O
22 X<>Y
23 MOD
24 ST- O
25 LASTX
26 ST/ O
27 CLX
28 X<> O
29 X<>Y
30 END
```

(59 bytes)

CHAPTER XVIII

UNIT CONVERSIONS & SHORTCUTS

18-1 CONVERT GRADS TO AND FROM DEGREES & RADIANS: pi rad = 180° = 200 grads.
G-D: .9, *. D-G: .9, /. G-R: 200, PI, /, /. R-G: 200, PI, /, *.

18-2 CONVERT FEET & INCHES TO FEET (OR YARDS): Input feet &/or inches in the form ff.ii (12 feet, 9 inches, for example, would be input as 12.09), then execute "FIN". The two steps before the RTN ('3, /') convert feet to yards--may delete.

LBL "FIN", ENTER, FRC, .12, /, X<>Y, INT, +, (3, /,), RTN (19 bytes)

18-3 APPROXIMATE MILES TO KILOMETERS CONVERSION FACTOR: To save bytes, use '5, LN'. This is about 0.0058% high (LN 5 = 1.609437912; the actual Mi-Km conversion factor is 1.609344000). Multiply miles (or MPH) by this factor to get kilometers (or KPH). Source: Neil Murphy (6) (65 NOTES, V1N2P3).

18-4 ENTERING ENGLISH OR METRIC UNITS, STORING AS METRIC: For use when values input are all positive (weight and height, for example). Enter data that is in English units as negative values, and enter data that is in metric units as positive values. In the program, have the English-to-metric conversion factor just before the prompting, and 'X>0?, 1, *' just afterwards. The example below prompts for weight (in pounds or kilograms), converts to kilograms if necessary, then stores the weight; similarly, it prompts for height (in inches or centimeters), converts to centimeters if necessary, then stores the height. Remember to follow English units with CHS. Source: Henry Casson (5047). The conversion factor must be negative.

..., -.4536, "WT.?", PROMPT, X>0?, 1, *, STO 01,
-2.54, "HT.?", PROMPT, X>0?, 1, *, STO 02, ...

18-5 FASTER ZERO: Use the decimal (.) rather than the zero (0) when a line in a program is to be zero; it's faster. This is only to enter a zero, not to clear X. It must not be followed by another digit. Source: Bill Kolb (265) (BP 67/97).

18-6 SYNTHETIC FASTER ONE: Use E (decimal byte 27) rather than 1 when a line in a program is to be one; it's faster.

18-7 MULTIPLY BY A SMALL NUMBER: Example: Usual--EEX, 6, CHS, *; better--EEX, 6, /. Saves a keystroke in keyboard execution, saves a byte in a program. Source: Bill Kolb (265) (BP 67/97).

18-8 DIVIDE BY 100: Instead of '100, /', use '1, %'. The number that is divided remains in the Y Register. Source: John Martellaro (1896) (PPC J, V5N1P16).

18-9 MULTIPLY BY $\sqrt{2}$: Instead of 2, SQRT, *, use ENTER, R-P. To replace x with $\sqrt{2}\cdot x$ without raising the stack, use X↑2, ST+ X, SQRT. Source: Bill Kolb (265) (BP).

18-10 4/3 PI: Usual: 4, PI, *, 3, /. Better: 240, D-R. Source: Bill Kolb (265) (BP).

18-11 $\pi/180$: Instead of PI, 180, /, use 1, D-R; saves 3 bytes. Source: John Martellaro (1896) (PPC J, V5N1P16).

18-12 1/(Y↑X): Usual: Y↑X, 1/X. Faster: CHS, Y↑X. Source: Bill Kolb (265) (BP67/97).

18-13 $y^{x/2}$: Usual: 2, /, Y↑X. Saves a byte: Y↑X, SQRT. Source: Bill Kolb (265) (BP).

18-14 CONVERSION BETWEEN DEGREES AND DECIMAL MINUTES & DECIMAL DEGREES ("DM-D" & "D-DM"): The Nautical Almanac uses degrees and decimal minutes. This routine prompts for input and labels output, so you won't be confused as to whether a decimal number is a decimal degree display, a degree-minute-second display, a degree-decimal minute display, or something else entirely. To convert from degrees and decimal minutes to decimal degrees, XEQ "DM-D" (input in DDDMM.MM format); to convert decimal degrees to degrees and decimal minutes, XEQ "D-DM". If routine is run as an independent program rather than as a subroutine in another program, just press R/S twice to convert back. Example: 123°45.67' = 123.7612°. Source: Hugh Kenner (103) (PPC CJ, V7N5P7, V7N6P35). See 18-16.

```
01 LBL "DM-D"
02 CF 29
03 "DDDMM.MM ?"
04 PROMPT
05 1 E2
06 /
07 ENTER
08 FRC
09 .6
10 /
11 X<>Y
12 INT
13 +
14 FIX 4
15 CLA
16 ARCL X
17 "├ DEG"
18 AVIEW
19 RTN
20 LBL "D-DM"
21 CF 29
22 "DEC DEG ?"
23 PROMPT
24 ENTER
25 INT
26 FIX 0
27 CLA
28 ARCL X
29 "├ D, "
30 1 E2
31 *
32 X<>Y
33 FRC
34 60
35 *
36 FIX 2
37 ARCL X
38 "├ M"
39 AVIEW
40 +
41 END
```

(109 bytes)

18-15 TO CONVERT NUMBERS TO ZERO OR ONE, DEPENDING UPON THEIR SIZE: This can save steps by avoiding conditional tests, branching and redundant operations. The same operation of addition or subtraction (ST+ 09, for example) can be used, and those numbers that become 1 will increment R09, while those that become 0 will not. To subtract 12 when the number becomes 1, but not when it becomes 0, multiply the 1 or 0 by 12 first. For larger entries to become 1, smaller entries to become 0: Divide the entry by the smallest number that is to yield 1; it and larger numbers will give a number greater than or equal to 1.0; then take INT. Smaller numbers give 0 after INT. If the range of possible inputs is too great, reduce it first by some method, such as taking the square root, or else test the result with 'X 0?, 1'. For small entries to become 1, large entries to become 0: Take the reciprocal, add an appropriate constant, then take INT. For example, if the range of possible inputs to a particular prompt is 1 - 100, and it is desired to convert entries of 10 or less to 1, but entries of greater than 10 to 0, use the following: '1/X, .9, +, INT'.

18-16 DDD.MM,M ⇌ DDD.DDD: Routine 01 converts degrees & minutes/tenths of minutes in DDD.MM,M format to decimal degrees; routine 02 converts the other way. Flag 02 must be clear before routine 01 is executed. Example: convert 123°45.67' to decimal degrees: 123.4567, XEQ 01; see 123.7612. Source: Bill Boulton (700) (PPC CJ, V8N1P22). See 18-14.

LBL 02, SF 02, LBL 01, INT, LASTX, FRC, .6, FS?C 02, 1/X, /, +, RTN (15 bytes)

18-17 CELSIUS-FAHRENHEIT TEMPERATURE CONVERSIONS ("TEMP"): With Flag 00 SET, this routine converts °C to °F; with Flag 00 CLEAR, it converts °F to °C. For example, 'SF 00, 50, XEQ "TEMP"' returns '122' (°F); 'CF 00, 50, XEQ "TEMP"' returns '10' (°C). Source: James Davidson (547) (65 NOTES, V3N9P14).

```
01 LBL "TEMP"
02 40
03 +
04 STO 01
05 1.8
06 ST* 01
07 /
08 40
09 ST- 01
10 -
11 FS? 00
12 RCL 01
13 RTN
```

(27 bytes)

18-18 CELSIUS-FAHRENHEIT CONVERSION, STACK SOLUTION ("TMP"): With Flag 00 SET, this routine converts the Celsius temperature in X to Fahrenheit; with Flag 00 Clear, it converts Fahrenheit to Celsius. No data registers are used. [continued]

Source: Ron Ryen (205) (65 NOTES, V2N5P9).

LBL "TMP", 40, +, 1.8, FC? 00, 1/X, *, 40, -, RTN (21 bytes)

18-19 CELSIUS-FAHRENHEIT CONVERSION, BOTH RESULTS ("TEM"): This routine converts an entry simultaneously to °C and °F; the Celsius value is returned to X, the Fahrenheit value is returned to Y. Uses no flags; uses no data registers. Source: Bill Derrick (1393) (PPC J, V5N7P4).

LBL "TEM", ENTER, ENTER, 32, ST- Z, X<>Y, 1.8, ST/ T, *, +, X<>Y, RTN (23 bytes)

18-20 INPUTTING A TEMPERATURE IN EITHER C° OR F° (SPECIAL CASE):

10°C = 50°F
10.56°C = 51°F
37°C = 98.6°F
50°C = 122°F

For photographic or clinical medical use, temperatures above 50°C are not met, so a value above that magnitude is bound to be Fahrenheit. The number could be varied for other uses. Assume any value keyed in that is less than or equal to 50 is a Celsius temperature, while any value keyed in that is greater than 50 is a Fahrenheit temperature.

CASE I: CONVERT TO CELSIUS: ..., 50, "TEMP?", PROMPT, X<=Y?, GTO 00, 32, -, 1.8, /, LBL 00, ...

CASE II: CONVERT TO FAHRENHEIT: ..., 50, "TEMP?", PROMPT, X>Y?, GTO 00, 1.8, *, 32, +, LBL 00, ...

Source: Cary Reinstein (2046) & Henry Casson (5047).

18-21 TEMPERATURE CONVERSIONS ("TC"): Uses no data registers, flags or numeric labels. Converts between any two of the following temperature scales: Celsius, Kelvin, Fahrenheit, and Rankine ('C', 'K', 'F' & 'R', respectively). To use: Input value to be converted; XEQ "TC" (ASN to E); press key (A-D) corresponding to value input; press key corresponding to desired output. See answer. For new case, key in new value, then press R/S or XEQ "TC"; repeat proceedure. Source: adapted from HP-41C Users' Library Solutions 'Heating, Ventilating & Air Conditioning', pp 69-72.

A	B	C	D	E
F	C	R	K	"TC"

```
01 LBL "TC"      07 +        13 +        19 PROMPT    25 GTO D     31 1.8
02 SF 27         08 1.8      14 GTO D    20 LBL A     26 LBL B     32 *
03 "F  C  R  K"  09 /        15 LBL C    21 1.8       27 273.15    33 LBL D
04 PROMPT        10 GTO D    16 1.8      22 *         28 -         34 END
05 LBL A         11 LBL B    17 /        23 459.67    29 GTO D
06 459.67        12 273.15   18 LBL D    24 -         30 LBL C     (96 bytes)
```

18-22 EFFECTIVE INTEREST: If you pay an income tax, this routine will calculate the approximate net cost of borrowing, taking into account the amount of interest you can deduct from your income tax:

LBL A, %, -, PSE, %, RTN

For example, you want to borrow $950 at 14.35% interest for one year, and your tax bracket is 38%. Key in 950, ENTER, 14.35, ENTER, 38; then press A. The display will pause to show '8.9' (your 'effective' interest), then stop to show '84.52' (your net cost of borrowing the $950). Remember, this is just a quick, handy way to determine approximate costs. It does not accurately calculate for direct reduction loans, compound interest, etc. But it is far better than nothing, it makes you more aware of net costs, and it is a short routine you can include in financial programs. Source: HP KEY NOTES, V4N2P11.

18-23 TWO NUMBERS WITHIN A CERTAIN PERCENT OF EACH OTHER: To find if the % difference between two numbers is less than a given number 'n' (perhaps in a loop), use the "%CH" function. Follow with "ABS", then 'n', then a conditional test.

CHAPTER XIX

STATISTICS & PROBABILITY

19-1 SUMMATIONS WITH FREQUENCY ("ΣF"): This routine allows for summations (Σ+ & Σ-) with frequency, so multiple sets of the same x,y pairs of values need only be entered once. "ΣF" sets ΣREG 04; it uses Flags 00, 21 & 27; minimum SIZE 010. Instructions: 1. XEQ "ΣF". 2. For i = 1, 2, ..., n, repeat the following: input x_i, ENTER, Y_i, ENTER, f_i; press A. (f = frequency.) 3. Correct a mistake by reentering x_i, y_i, f_i, then pressing C. 4. Press E for intermediate or final results. If a printer is not on line, press R/S between outputs. 5. To add more data, go to step 2. 6. For a new case, go to step 1.

Example:

x	1	2	4	6
y	1	3	5	7
f	15	2	2	1

Results:

ΣX= 33.00   ΣY↑2=132.00
ΣX↑2=91.00   ΣXY=109.00
ΣY=38.00   N=20.00

Note: After LBL E, other calculations and output instructions can be inserted. For example, to find the means of x and y, insert these steps: 'MEAN, "XBAR", XEQ 02, X<>Y, "YBAR", XEQ 02.

Source: adapted from 'Basic Statistics for Two Variables', HP-41C Stat Pac, pp 10-13.

A	INPUT		CORRECT		RESULTS	E

```
01 LBL "ΣF"
02 ADV
03 9
04 STO 00
05 CLX
06 LBL 01
07 STO IND 00
08 DSE 00
09 GTO 01
10 CF 00
11 SF 21
12 SF 27
13 ΣREG 04
14 "X,↑,Y,↑,F: A"
15 PROMPT
16 LBL 00
17 "PRESS A OR C"
18 PROMPT
19 LBL C
20 SF 00
21 LBL A
22 STO 03
23 FS? 00
24 CHS
25 ST+ 09
26 RDN
27 STO 02
28 RDN
29 STO 01
30 R↑
31 R↑
32 ABS
33 *
34 *
35 FS? 00
36 CHS
37 ST+ 08
38 RCL 02
39 X↑2
40 RCL 03
41 *
42 FS? 00
43 CHS
44 ST+ 07
45 RCL 02
46 RCL 03
47 *
48 FS? 00
49 CHS
50 ST+ 06
51 RCL 01
52 X↑2
53 RCL 03
54 *
55 FS? 00
56 CHS
57 ST+ 05
58 RCL 01
59 RCL 03
60 *
61 FS?C 00
62 CHS
63 ST+ 04
64 RCL 09
65 STOP
66 GTO 00
67 LBL E
68 RCL 04
69 "ΣX"
70 XEQ 02
71 RCL 05
72 "ΣX↑2"
73 XEQ 02
74 RCL 06
75 "ΣY"
76 XEQ 02
77 RCL 07
78 "ΣY↑2"
79 XEQ 02
80 RCL 08
81 "ΣXY"
82 XEQ 02
83 RCL 09
84 "N"
85 LBL 02
86 "⊢="
87 ARCL X
88 AVIEW
89 END
```

REGISTERS:

R00: Pointer

R01: x_i

R02: y_i

R03: f_i

R04: Σx

R05: Σx^2

R06: Σy

R07: Σy^2

R08: Σxy

R09: n

(173 bytes)

19-2 RECIPROCAL OF SUMS OF RECIPROCALS ("ΣRECIP"): Data can be entered in any order and the intermediate answer can be seen at any time. Example:

$$R = \frac{1}{\Sigma S + \sum\frac{1}{T} + \sum\frac{U}{2} + \sum\frac{V}{W}}$$

This routine must be rewritten for different equations; the listing below is for this equation. No numeric data registers are used (stack solution).

Instructions: 1. XEQ "ΣRECIP" (ASN to E). Note the top row of keys are now defined in the display. 2. To add an 'S', input S, press A; to add a '1/T', input T, press B; to add a 'U/2', input U, press C; to add a 'V/W', input V, ENTER, W, press D. 3. Repeat step 2 until ready for an answer. 4. R/S for intermediate or final results. 5. R/S again (if no printing) to add more data to the intermediate sum; go to step 2. 6. For a new case, go to step 1. 7. After keying a new value, if you forget which key to press, briefly switch to Alpha Mode to see the prompt again, then switch back and press appropriate key.

A						E
	S	T	U	V,↑,W	"ΣRECIP"	

```
01 LBL "ΣRECIP"
02 LBL E
03 ADV
04 SF 21
05 SF 27
06 0
07 LBL 01
08 " S T  U V↑W"
09 PROMPT
10 " R="
11 ARCL X
12 AVIEW
13 GTO 01
14 LBL A
15 "S"
16 XEQ 00
17 GTO 02
18 LBL B
19 "T"
20 XEQ 00
21 1/X
22 GTO 02
23 LBL C
24 "U"
25 XEQ 00
26 2
27 /
28 GTO 02
29 LBL D
30 X<>Y
31 "V"
32 XEQ 00
33 X<>Y
34 "W"
35 XEQ 00
36 /
37 LBL 02
38 X<>Y
39 X≠0?
40 1/X
41 +
42 1/X
43 GTO 01
44 LBL 00
45 "├="
46 ARCL X
47 SF 25
48 PRA
49 CF 25
50 END
```

(109 bytes)

19-3 LAST X MAY NOT BE SAVED WITH SUMMATIONS: Early HP-41C's (up to about serial number 1938A2000) had a LASTX 'bug': LAST X wasn't saved using "Σ+" or "Σ-". If your machine has this bug, follow the Σ+ or Σ- instruction with "STO L". The bug is eliminated with routine ROM updating when serviced. Source: Bill Kolb (265).

19-4 SYNTHETIC SIGMA FINDER ("Σ?"): XEQ "Σ?" to find the number of the first register of the statistics block. See routine 1-17. Source: Keith Jarett (4360) & Roger Hill (4940) (PPC ROM).

```
01 LBL "Σ?"
02 CLA
03 XEQ "C?"
04 RCL N
05 XEQ 14
06 CLA
07 X<>Y
08 -
09 RTN
10 LBL "C?"
11 RCL c
12 LBL 14
13 STO M
14 "├◆◆A"
15 X<> M
16 X<> d
17 CF 01
18 CF 02
19 CF 04
20 CF 07
21 FS?C 10
22 SF 07
23 FS?C 11
24 SF 09
25 FS?C 12
26 SF 10
27 FS?C 13
28 SF 11
29 FS?C 14
30 SF 13
31 FS?C 15
32 SF 14
33 FS?C 16
34 SF 15
35 X<> d
36  E38
37 /
38 INT
39 DEC
40 END
```

(87 bytes)

19-5 SYNTHETIC RECALL SIGMA ("RΣ"): This routine duplicates the HP-67/97 RCLΣ function by returning Σx to X and Σy to Y. Useful for resolution of forces. It uses the Synthetic Sigma Finder Routine ("Σ?") above (19-4). Source: John Dearing (2791).

LBL "RΣ", XEQ "Σ?", 2, X<>Y, +, RCL IND X, RCL IND L, RTN (18 bytes)

19-6 SIGMA RECALL ("ΣR"): This routine replaces the values in Y and X with ΣY and ΣX, respectively, from the summation registers; this simulates the RCLΣ function of other calculators. The values in Z and T are left unchanged; 'n' is returned to L (the LASTX Register). The values in the summation registers are left unchanged, and this routine works for any location of the summation registers; 'n' must not be zero. This routine doesn't need a sigma-find subroutine, so it executes in just over a second. Source: Jurgen K. Cappel (6015) (PPC CJ, V8N2P16).

```
01 LBL "ΣR"    03 STO Y    05 0     07 MEAN     09 ST* L    11 X<> L
02 CLX         04 Σ+       06 Σ-    08 X<> L    10 ST* Y    12 END       (24 bytes)
```

19-7 MANUAL SUM OF X- AND Y-VALUES ("XYΣ"): To manually find the sum of x- and y-values using the stack, this routine will help. To use: 1. XEQ "XYΣ" to initialize (to clear the stack). 2. Key in y, ENTER, x; R/S. 3. Repeat step 2 as desired. 4. See Σx in X; X<>Y to see Σy. 5. To add more data pairs, X<>Y to return Σx to X and Σy to Y, then go to step 2. Source: John Dearing (2791).

```
LBL "XYΣ", CLST, LBL 14, ST+ Z, RDN, ST+ Z, RDN, STOP, GTO 14      (18 bytes)
```

19-8 BLOCK PLUS ("BP"): This routine will calculate the sum of the values in the block of registers defined by a control number (bbb.eee) in R00. Set or clear Flag 10 as desired, then XEQ "BP". IF FLAG 10 IS SET, the sum of the values in the block of registers (Σx) is returned to the first register of the statistics block, defined by the ΣREG function (default: R11). Also, the sum of the squares of the values (Σx^2) is returned to the next higher register (default: R12), and the number of registers in the block (n) is returned to the register that is 5 greater than the first register of the statistics block (default: R16). X will be cleared. IF FLAG 10 IS CLEAR, the sum of the values in the block is returned to X, and also to R'eee+1'. In both cases, the contents of Registers bbb - eee are unchanged. Source: PPC CJ, V7 N10P7; Richard Nelson (1).

```
01 LBL "BP"    04 RCL IND 00    07 FS? 10    10 +         13 FC? 10
02 0           05 FS? 10        08 CLX       11 ISG 00    14 STO IND 00
03 LBL 06      06 +             09 FC? 10    12 GTO 06    15 END          (30 bytes)
```

19-9 LITTLE BLOCK PLUS ("B+"): To return the sum of the values in a block of registers to X, key in the bbb.eee control number defining the block, then XEQ "B+". No flags are used, no data registers are altered.

```
LBL "B+", 0, LBL 06, RCL IND Y, +, ISG Y, GTO 06, RTN      (16 bytes)
```

19-10 SYNTHETIC STATISTICS BLOCK ("ΣB"): INPUT: bbb.eee control number in X, defining the block. OUTPUT: Σx in X, Σx^2 in Y, n in Z, & $\bar{x}$ in T. Uses Alpha Register. Source: John Dearing (2791).

```
01 LBL "ΣB"    05 ""            09 ST+ O     13 RCL M     17 RCL O
02 CLA         06 RCL IND X     10 RDN       14 RCL N     18 RCL M
03 LBL 08      07 ST+ M         11 ISG X     15 /         19 END
04 ISG N       08 X↑2           12 GTO 08    16 LASTX                  (36 bytes)
```

19-11 BLOCK STATISTICS ("BΣ"): With a bbb.eee control number defining a block of y-values in Y, and the number of the first register of a block of x-values in X, XEQ "BΣ" to return the usual Σx, Σx^2, Σy, Σy^2, Σxy, and n to the statistics register block defined by the ΣREG function. With carefully-selected control numbers, the x- and y-values can be in adjacent registers; for example, input 2.00802, ENTER, 1.00002, XEQ "BΣ"; the routine will treat as y-values the contents of R02, R04, R06 & R08; it will treat as x-values the contents of R01, R03, R05 & R07. This routine may be considered to be a matrix routine since it can be used to compute vector dot products. Given the appropriate input parameters, this routine can be used to compute matrix products (to multiply a row in one matrix by a column in another matrix). Source: John Kennedy (918) & Richard Schwartz (2289) (PPC ROM).

```
01 LBL "BΣ"    04 RCL IND Y    07 R↑       10 STO X     13 RTN
02 CLΣ         05 RCL IND Y    08 R↑       11 ISG Y
03 LBL 12      06 Σ+           09 ISG X    12 GTO 12                (23 bytes)
```

19-12 PERMUTATIONS & COMBINATIONS ("PERM" & "COMB"): "PERM" will compute the number of permutations of 'n' objects taken 'k' at a time: P(n,k) = n! / [(n-k)!]. Input n, ENTER, k, XEQ "PERM". Example: P(73,4) = 26,122,320. "COMB" will compute the number of combinations of 'n' objects taken 'k' at a time: C(n,k) = n! / [k!(n-k)!]. Input n, ENTER, k, XEQ "COMB". Example: C(73,4) = 1,088,430. Source: Jim Davidson (547) & Bill Derrick (1393) (PPC J, V5N7P3).

```
01 LBL "PERM"
02 STO 00
03 X<>Y
04 STO 01
05 DSE 00
06 LBL 14
07 RCL 01
08 RCL 00
09 -
10 *
11 DSE 00
12 GTO 14
13 RTN
14 LBL "COMB"
15 STO 00
16 X<>Y
17 STO 01
18 1
19 LBL 13
20 RCL 01
21 RCL 00
22 /
23 *
24 1
25 ST- 01
26 *
27 DSE 00
28 GTO 13
29 END
 (51 bytes)
```

19-13 COMPACT PERMUTATIONS & COMBINATIONS ("PRM", "COM" & "P+C"): These routines are compact, fast, and use no data registers, but they will not work if either 'n' or 'k' is greater than 69. For each routine, input n, ENTER, k, then execute the routine. "PRM" returns the permutation to X, "COM" returns the combination to X, and "P+C" returns both--the permutation to X, the combination to Y. Examples: P(9,4) = 3024; C(9,4) = 126. Source: Chris Stevens (3005) (PPC CJ, V7N5P44).

```
01 LBL "PRM"
02 LBL 12
03 X<>Y
04 FACT
05 LASTX
06 RCL Z
07 -
08 FACT
09 /
10 RTN
11 LBL "COM"
12 XEQ 12
13 X<>Y
14 FACT
15 /
16 END
 (33 bytes)
```

```
01 LBL "P+C"
02 X<>Y
03 FACT
04 RCL Y
05 ST- L
06 X<> L
07 FACT
08 /
09 RCL Y
10 FACT
11 1/X
12 X<>Y
13 ST* Y
14 END
 (27 bytes)
```

19-14 STACK PERMUTATIONS & COMBINATIONS ("PM" & "CM"): P(n,k) is the number of permutations of 'n' objects taken 'k' at a time and is the number of arrangements or orderings of all subsets of size k selected from a set of n objects. C(n,k) is the number of combinations of 'n' objects taken 'k' at a time and is simply the number of subsets (order doesn't matter) of size k selected from a set of n objects. Input for both is n, ENTER, k. "PM" saves Z & T in Y & Z; "CM" saves Y in Y. No data registers are used. Source: John Kennedy (918) & Keith Jarett (4360) (PPC ROM).

```
01 LBL "PM"
02 CHS
03 X<>Y
04 SIGN
05 X<> L
06 ST+ Y
07 LBL 06
08 X=Y?
09 GTO 07
10 ST* L
11 DSE X
12 GTO 06
13 LBL 07
14 RDN
15 X<> L
16 RTN
17 LBL "CM"
18 RCL Y
19 RCL Y
20 X≠Y?
21 -
22 X>Y?
23 X<>Y
24 ST+ T
25 SIGN
26 X<>Y
27 LBL 08
28 X<> T
29 LASTX
30 ST- Y
31 /
32 ST* Y
33 DSE L
34 GTO 08
35 RDN
36 END
 (63 bytes)
```

19-15 EASY POPULATION STANDARD DEVIATION (σ): After accumulations, execute 'MEAN, Σ+, SDEV'. σ_x is returned to X and σ_y is returned to Y. To restore the statistics registers, use 'MEAN, Σ-'. Source: Joseph Horn (1537) (PPC CJ, V7N4P13).

CHAPTER XX

TIME & DATE

20-1 JULIAN DAY NUMBER ("JD"): This is a calendar routine which computes the Julian Day Number of a given date. The valid range is from September 14, 1752. Only the stack is used. Key in the date in X in mm.ddyyyy format before executing; the Julian Day Number is returned to X. Source: Fred Wheeler (1150) (PPC CJ, V7N8P11).

```
01 LBL "JD"
02 FRC
03 3
04 LASTX
05 INT
06 1
07 +
08 X>Y?
09 GTO 07
10 +
11 1 E-6
12 ST- T
13 RDN
14 9
15 +
16 LBL 07
17 30.6
18 *
19 INT
20 1 E2
21 R↑
22 *
23 ENTER
24 INT
25 ST+ Z
26 -
27 1 E2
28 *
29 INT
30 CHS
31 36525
32 LASTX
33 *
34 ST+ Z
35 X<> L
36 4
37 /
38 INT
39 1720997
40 +
41 +
42 +
43 INT
44 END
```

(76 bytes)

20-2 CALENDAR DATE / JULIAN DATE CONVERSIONS ("CJ" & "JC"): "CJ" (Calendar Date to Julian Day Number): This routine computes the Julian Day Number of a given day with a valid range from March 1, year 0. With Flag 10 clear, input Gregorian calendar dates; with Flag 10 set, input Julian calendar dates. The input is of the form with the year in Z, the month in Y and the day in X. "JC" (Julian Day number to Calendar Date): This routine is the inverse of "CJ"--it computes a calendar date, given the Julian Day Number of the date. Input: Julian Day Number in X; output: the year in Z, the month in Y, and the day of the month in X. With Flag 10 clear, the output date is for the Gregorian calendar; with Flag 10 set, the output date is for the Julian calendar. These routines use no numeric data registers. Source: Kennedy (918), Wheeler (1150) & Hill (4940) (PPC ROM).

```
01 LBL "CJ"
02 INT
03 X<>Y
04 INT
05 2.85
06 -
07 12
08 /
09 R↑
10 INT
11 +
12 X<0?
13 SQRT
14 ENTER
15 INT
16 ST- Z
17 X<>Y
18 367
19 *
20 INT
21 ST+ Z
22 SIGN
23 FS? 10
24 ISG X
25 %
26 INT
27 .75
28 ST* Z
29 *
30 RDN
31 -
32 INT
33 R↑
34 -
35 INT
36 1721115
37 +
38 RTN
39 LBL "JC"
40 INT
41 1721119.2
42 -
43 ENTER
44 FS? 10
45 -2
46 FS? 10
47 GTO 09
48 36524.25
49 /
50 INT
51 ST+ Y
52 4
53 /
54 INT
55 LBL 09
56 -
57 X<0?
58 SQRT
59 STO Y
60 365.25
61 ST/ Y
62 X<>Y
63 INT
64 ST* Y
75 RDN
66 INT
67 -
68 .3
69 -
70 STO Y
71 30.6
72 ST/ Y
73 X<>Y
74 INT
75 *
76 ST- Y
77 ISG Y
78 X<> L
79 -3
80 X↑2
81 X<Y?
82 ISG T
83 X<> L
84 -
85 X<>Y
86 INT
87 END
```

(158 bytes)

20-3 STOPWATCH ("TM"): As prompted, key in the current time in HH.MMSS format and press R/S. If too fast, slightly decrease the number in Line 06; if too slow, increase the number. NOTE: The 41C/V times with an oscillator, not a crystal; this routine can be fine-tuned for a given calculator under given conditions, but accuracy cannot be guaranteed. Stop the timer with R/S; the time will be in the X or the

Y Register. Source: HP KEY NOTES, V4N3P11.

```
01 LBL "TM"
02 "TIME, HH.MMSS?"
03 PROMPT
04 FIX 4
05 CF 21
06 LBL 00
07 .000057
08 HMS+
09 VIEW X
10 GTO 00
11 RTN
(42 bytes)
```

20-4 PRINT CALENDAR ("PC"): To use: 1. Set SIZE 004 or greater. 2. XEQ "PC". 3. as prompted, input starting year, starting month, and number of months to print. For starting month, if 0 is entered, or if no entry is made, then the next prompt is skipped and the entire year is printed. Valid for any year. Source: Roger Hill (4940) (PPC CJ, V7N6P14).

```
01 LBL "PC"
02 "START YEAR?"
03 PROMPT
04 INT
05 X<=0?
06 LN
07 STO 01
08 .23
09 -
10 STO 00
11 CLX
12 "MONTH?(0-12)"
13 PROMPT
14 ABS
15 INT
16 ENTER
17 X=0?
18 SIGN
19 STO 02
20 12
21 STO 03
22 X<Y?
23 ASIN
24 /
25 ST+ 00
26 X>Y?
27 GTO 10
28 SIGN
29 "NO. MONTHS?"
30 PROMPT
31 INT
32 STO 03
33 LBL 10
34 PRBUF
35 FIX 0
36 CF 29
37 .012
38 ST+ 02
39 31
40 RCL 00
41 *
42 INT
43 RCL 00
44 INT
45 STO 00
46 1
47 %
48 INT
49 .75
50 ST* 00
51 *
52 +
53 -
54 INT
55 RCL 00
56 -
57 INT
58 3
59 +
60 7 E-5
61 +
62 STO 00
63 LBL 20
64 SF 12
65 CLX
66 ADV
67 XEQ IND 02
68 GTO 13
69 LBL 01
70 "JAN"
71 RTN
72 LBL 03
73 "MAR"
74 RTN
75 LBL 05
76 "MAY"
77 RTN
78 LBL 07
79 "JUL"
80 RTN
81 LBL 08
82 "AUG"
83 RTN
84 LBL 10
85 "OCT"
86 RTN
87 LBL 12
88 "DEC"
89 RTN
90 LBL 04
91 "APR"
92 SIGN
93 RTN
94 LBL 06
95 "JUN"
96 SIGN
97 RTN
98 LBL 09
99 "SEP"
100 SIGN
101 RTN
102 LBL 11
103 "NOV"
104 SIGN
105 RTN
106 LBL 02
107 "FEB"
108 SIGN
109 RCL 01
110 %
111 ENTER
112 INT
113 X≠Y?
114 RCL 01
115 4
116 MOD
117 X≠0?
118 SIGN
119 2
120 +
121 RTN
122 LBL 13
123 ACA
124 RCL 01
125 ACX
126 2
127 SKPCHR
128 ADV
129 CF 12
130 24
131 R↑
132 -
133 RCL 00
134 7
135 MOD
136 +
137 STO 00
138 " SU MO TU WE"
139 "├ TH FR SA"
140 PRA
141 3
142 LASTX
143 *
144 SKPCHR
145 " "
146 ACA
147 9 E-3
148 SF 10
149 GTO IND L
150 LBL 14
151 GTO IND Z
152 LBL 00
153 ISG X
154 ACA
155 ACX
156 LBL 01
157 ISG X
158 ACA
159 ACX
160 LBL 02
161 ISG X
162 ACA
163 ACX
164 LBL 03
165 ISG X
166 ACA
167 ACX
168 LBL 04
169 ISG X
170 ACA
171 ACX
172 LBL 05
173 ISG X
174 ACA
175 ACX
176 LBL 06
177 ISG X
178 ACA
179 ACX
180 PRBUF
181 ACA
182 DSE Z
183 GTO 00
184 FS?C 10
185 GTO 14
186 ISG 02
187 GTO 10
188 12
189 ST- 02
190 ISG 01
191 LBL 10
192 DSE 03
193 GTO 20
194 END
(367 bytes)
```

```
 JUN  1982
SU MO TU WE TH FR SA
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

 JUL  1982
SU MO TU WE TH FR SA
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

 AUG  1982
SU MO TU WE TH FR SA
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

 JAN  2000
SU MO TU WE TH FR SA
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

CHAPTER XXI

CARD READER & WAND

21-1 CARD READER CURRENT DRAIN AND WEAR: Keep plugging and unplugging the 82104A Card Reader to a minimum. Keep the card reader in the machine when not in use. Cycle on/of after plugging in. When first plugged in, the card reader may draw excessive current. Turn on, then off, to reduce to normal. Leaving the card reader plugged in as much as possible helps prevent electrostatic charge crashes, contact wear on port block metalized plastic contact surfaces, and helps keep dust and foreign objects out of ports. Source: Philip Karras (3480) (PPC CJ, V7N2P56).

21-2 COMPLETING CARD READS: Preserve batteries by immediately completing pending card reads instead of having a cup of coffee and leaving the HP-41 with "RDY NN OF NN" in the display. Source: Richard Nelson (1) (PPC J, V6N7P4).

21-3 CARD STUCK BECAUSE OF VERY LOW BATTERIES: Normally, when the N cells are getting weak and one tries to read/write a card, the 82104A Card Reader will warn you and then pass the card on through without a read/write. But, if you forget that your batteries are very weak (though still able to run the calculator) and you try to use the card reader, the following may happen: warning is given, and the card starts through, but slows down to a grinding halt. The card is held quite firmly, and it might damage the mechanism to force it through or try to pull it back out. The calculator stays on, displaying "LOW BAT", and cannot be turned off! Suggested procedure to correct: 1: Unplug the card reader. 2: Turn the calculator OFF. 3: Plug in a new set of cells. 4: Plug in the card reader--card will complete its pass. 5: Turn calculator ON and continue with another try at read/write. Source: Fred Wheeler (1150) (PPC CJ, V7N3P27).

21-4 BEHAVIOR OF "RSUB" & "MRG": The card reader functions "RSUB" and "MRG" work differently than the manual states: RSUB will replace the last program only if it does not have an END; otherwise it will be placed after the last END. MRG will show "MRG ERR" if the calculator is not placed at the last program, but will still read the card and place it after the last END. Source: Dennis Green (4213) (PPC CJ, V7N3P27).

21-5 WHAT TO DO IF A CARD WON'T READ: Try breathing on its magnetic (dark) surface first. Source: John Burkhart (4382). Try rubbing it on your shirt. Source: Henry Casson (5047). Try cleaning it with a touch of rubbing alcohol and a soft cloth. Also be aware that if magnetic cards touch or come near anything that is magnetized, they will be ruined. Source: Bill Kolb (265).

21-6 ENTERING DATA AS PART OF THE PROGRAM: In many program usage situations, it is confusing to have both Program and Data cards. Decreased magnetic card usage and simpler user instructions often result if the 'data' is entered as part of the program. Flag 11 (auto execution) can be set when the program is recorded, and the first part of the program can automatically execute to store the data into the registers. This method of combining data and program greatly simplifies program usage. Source: HP KEY NOTES, V7N2P7.

21-7 STATUS CARD AS FIRST CARD OF A PROGRAM SET: A Status card can be used to automatically set the Size (gives "SZE ERR" message if there's not enough room), status of Flags 00 - 43, ΣREG location, and stack and Alpha Register contents, all on Track 1. [Subsequent track(s) of status card(s) can contain key assignments.] An appropriate alpha message (6 characters or fewer) can be placed in the X Register (as "NEXT"), to be seen when the card is read; a longer (up to 24-character) message could be put in the Alpha Register, and "ALPHA" put in the X Register as a prompt to press the ALPHA key to see the message. Source: HP KEY NOTES, V4N2P7.

21-8 CLEAR ASSIGNMENTS CARD: To make: Master Clear; make, then delete one key assignment; then XEQ "WSTS", reading in one track of the card twice to record only track 2 (rerecording track 1 so that only track 2 is retained). To use: Read the track; backarrow (press correction key) to clear the "RDY 01 OF 02" prompt; then turn the calculator OFF, then ON, to regain the cleared status registers. Source: Roger Hill (4940) (PPC CJ, V7N8P22). See 1-26.

21-9 HANG-UP WITH "SIZE", "DEL" OR "GTO.": When a card reader is plugged in, if you execute SIZE, DEL or GTO., key 2 digits and then try to backarrow twice to remove both digits, the 41C/V may hang up. Pressing any key except the backarrow key will recover, with the loss of the function prompt. This is a bug of the card reader, not the calculator, as can be verified by removing the card reader and trying again. Source: Bill Kolb (265).

21-10 'LAST PROGRAM' IN THE CARD READER CONTEXT: The last program, in the card reader context, consists of whatever follows the last END in memory. If the final program terminates with an END, then it is not the last program in this sense. The 'last program' is then the section containing "00 REG nn, .END. REG nn". Hence, for example, if the final keyed program contains an END, you can't merge a program on to it. Note that the card reader does not record END statements; this is useful in light of the above. Source: Bill Wickes (3735).

21-11 READING PART OF A "WALL" SET OF CARDS: Contrary to the manual, a "WALL" set of cards does not have to be entered in its entirety. You may stop the read process at any time by momentarily removing the batteries, replacing the batteries, and then pressing the backarrow (correction) key. Use this technique to reclaim key assignments, and programs, provided there is ample program space available to overlay the desired program. Source: Bill Kolb (265).

21-12 RESUMING PROGRAM EXECUTION AFTER A "WALL": "WALL" records program pointer position and RTN addresses, enabling interruption and resumption of running programs. Source: Bill Wickes (3735).

21-13 KEY ASSIGNMENTS CARD REMINDER: When recording a special set of key assignments, give the card a name that will help you remember what is on it. Key this name (or the key assignment mneumonics) into the Alpha Register prior to recording the status. If ALPHA Mode is on when the cards are read, the name will appear in the display. If not, simply switch to ALPHA to read the names. A short name can be ASTO'd into X as a similar reminder. Source: Bill Kolb (265).

21-14 KEYING FUNCTIONS OF MISSING PERIPHERALS: If the printer, wand or card reader isn't connected, the built-in functions can still be called by simply spelling out the function name. For example, to obtain "PRX" in a program when a printer is not plugged in, key in 'XEQ, ALPHA, P, R, X, ALPHA'. When executed in the program, the calculator will search for a user program named "PRX"; if none is found, it will execute PRX on the printer, if the printer is plugged in and is on. Source: Bill Kolb (265).

21-15 TO PUT "VER" OR "WPRV" INTO A PROGRAM: 1. With the card reader in place, ASN VER or WPRV to a key. 2. To insert them into a program, turn the HP-41 OFF, remove the card reader, turn ON, set PRGM Mode, press the assigned key. The line will read "XROM 30,05" or "XROM 30,09", respectively. 3. Turn the HP-41 OFF, reattach the card reader, turn ON, set PRGM Mode. The line will now read VER or WPRV and will execute as such. Synthetic Method: The same result can be reached without removing the card reader, by creating these functions synthetically: "VER" = decimal 167, 133 (hex A7 85); "WPRV" = decimal 167, 137 (hex A7 89).

Follow WDTAX in a program with VER to check whether the writing was successful. The example below loads each register from R00 to R16 with its own address, then executes "WDTAX". As prompted, insert both tracks of a blank card to record this data; when all data is written, the prompt will automatically change to "CARD" (for VER); read the same tracks to verify that WDTAX executed properly. If it did (you see "TYPE D, TR 01", etc), press R/S twice (or backarrow, R/S) to continue program execution. If it did not (the VER test fails, and you see "CARD ERR"), press backarrow to clear the prompt, then 'SHIFT, BST' twice to the WDTAX instruction, and then R/S to try again. Source: Cary Reinstein (2046) and John Herzfeld (5428).

01 LBL "X"	05 LBL 14	09 ISG Y	13 VER	17 CLD
02 .016	06 STO IND Y	10 GTO 14	14 "MORE STEPS HERE"	18 END
03 ENTER	07 ISG X	11 LASTX	15 AVIEW	
04 INT	08 LBL 00	12 WDTAX	16 PSE	(48 bytes)

21-16 INTERFERENCE AND THE WAND: If you have problems using your wand and especially if you notice that it seems to work better on some days than on others, you may be experiencing some form of interference from the AC power line. If you hold the wand below the on/off switch, your body may couple AC powerline 60 Hz 'hum' into the wand. This is possible because of the very high gain circuits that must process the very low level optical signal due to the reflected light from the paper. If strong AC signals get coupled into the wand, it will cause a nonread condition and you won't get any response from the machine. This may happen if the 41C/V is close to a transformer, electrical wiring, fluorescent lights, etc. The printer acts as an antenna when it is plugged in and may increase the interference because it and its cable add to the problem. The following tips will help if you suspect the wand is being affected by interference: a. Hold the wand above the bottom of the on/off switch. Most users do this naturally. b. If electrical interference is suspected, operate the wand without other system peripherals connected, especially those that may plug into the AC line, or move the system to another location. NOTES: 1. If there's a question of proper performance, test by using the Paper Keyboard instructions: scan several times. Out of 25 scans, you should get at least 23 'good' responses. Poor quality barcode will increase the sensitivity to interference. 2. Move to another location if interference is suspected. Working on a light table or display case may be an interference situation because of the fluorescent lights. Moving the calculator just two inches may be a big help if you are very close to the source. 3. A quick test for 'good reads' is to scan the "+" instruction, with the Y, Z & T Registers loaded with '1' and '0' in X. Scan 25 times. The display will show the number of good reads. Interference can cause the good reads to drop from 24 or 25 to 12 - 16. Interference should seldom be a problem if you know how to recognize it and isolate the system from it. Source: Richard Nelson (1) (PPC CJ members letter, June 1980).

21-17 WAND TIPS: Photocopies of photocopies of barcodes often won't read: try rotating the first copy 90 or 180° before copying. Always use a sheet protector. non-glare preferred, so that photocopies won't smear, and to keep copies clean. Do not tightly coil the wand cable for handling or storage, in order to prevent 'kinky' cable from getting in the way. Barcodes printed on light-weight paper may 'bleed through' and be seen by the wand, so place a black sheet behind the printed sheet. If the wand is used interactively with the HP-41, it might be a good programming practice to turn the calculator OFF when the program is finished; this will save power, and since the wand will turn the HP-41 ON again, there is little inconvenience. Include data checking in your program if you use WNDSCN to read simple 'positioned' data. This is a good practice because the wand does no checking of the bars it reads using WNDSCN. Even a simple out of range check is better than none. Use two tone 9's to prompt to scan a new set of bars; use two tone 89's if in a hurry. Only WNDTST uses 'HP-41' Code; the other five functions, WNDDTA, WNDDTX, WNDLNK, WNDSUB and WNDSCN, are microcode functions and cannot be listed. Source: Richard Nelson (1) (PPC CJ, V7N5P22).

CHAPTER XXII

PRINTER

22-1 PRINT PROMPT & INPUT: VIEW and AVIEW cause the display to be printed if the printer is plugged in and turned on (Flags 21 & 55 set). Tip--instead of using PROMPT to stop for an input after an alpha message, use AVIEW, STOP: the message will be printed. Follow with VIEW X to print the input right justified, or follow with CLA, ARCL X, AVIEW to print the input left justified. Source: HP KEY NOTES, V3 N4P4. See 4-18, 6-14.

22-2 PAPER OUT ("PO"): When desired after printing, XEQ "PO" to advance paper to the point where it can be torn off. Source: PPC ROM.

LBL "PO", ADV, ADV, ADV, ADV, ADV, RTN (12 bytes)

22-3 PRINTER TIPS:

Register 06: R06 is the register to use when entering a function using X more than once, when using PRPLOT to print graphs.

NORMal Mode: Numbers and alpha strings are printed as keyed in, and function names are printed as executed from keyboard; all print functions print. (Example of use: checkbook balancing).

PROMPT: Prints the Alpha Register in NORM and TRACE Modes.

BATTERY PACK: CAUTION: the battery pack must be in the printer while the AC adapter/recharger is connected. The printer may be damaged otherwise.

CLASSIFICATION OF PRINTER FUNCTIONS:

Stack & Alpha:	Data:	Program:	Status:	Plotting:	Buffer:	Graphics:
PRX & VIEW	PRREG	PRP	PRFLAGS	"PRPLOT"	ACA	ACCOL
PRA & AVIEW	PRREGX	LIST	PRKEYS	"PRPLOTP"	ACX	SKPCOL
PRSTK	PRΣ		CAT 1†	"PRAXIS"	ACCHR	SKPCHR
				REGPLOT	ASKCHR	BLDSPEC
		(†in Trace Mode)		STKPLOT	PRBUF	ACSPEC
					ADV	

PROGRAMMING & THE PRINTER: In TRACE or NORM, each line of a program is printed as you key it in. Executing a program while in TRACE Mode prints what's going on in the program, but uses a lot of paper.

EXECUTING A PROGRAM WITH NO PRINT FUNCTIONS: Turn the printer off to have the program execute slightly faster, as the HP-41 always interacts with the printer during program execution if the printer is on, slowing program execution.

FLAG 55: Always set if the printer is plugged in, whether it's on or off.
FLAG 21: If clear, supresses printing, even if the printer is on.

PRINT WIDTH: Maximum print width = 24 characters = 168 columns. ACCOL allows you to print special graphics up to 43 columns wide; SKPCOL skips X columns, up to 167.

PRINT & ADVANCE: Pressing PRINT while the 41C/V is in PRGM Mode inserts a PRX into the program; or, if also in Alpha Mode, it inserts PRA instead. Pressing PAPER ADVANCE in PRGM Mode inserts an ADV into the program (unless the PAPER ADVANCE key is held down longer than a second, in which case ADV is not inserted into the program, but rather the paper is immediately advanced).

PRINTING CATALOGS: Set the print mode switch to TRACE before executing CAT to print the specified catalog. If CAT 1 is printed, the number of bytes each program occupies in program memory will also be printed.

CLEARING BUFFER WITHOUT PRINTING: To clear the printer buffer without printing, turn the printer off, then on again.

PROGRAMS CONTAINING ACCUMULATION FUNCTIONS should not be executed in TRACE or NORM, as these modes use the buffer registers, and hence will cause the buffer to print prematurely.

PRINTER PLOTTING: 1. The name of the program prompted for by "NAME?" must be 6 or fewer characters. 2. When prompted by "X INC?", a positive response indicates the X-increment, while a negative response indicates the number of X-increments. 3. When prompted by "AXIS?", any Alpha input yields a graph with no axis printed. 4. If Register 03 contains an output generated by BLDSPEC, that character will be the plot character; otherwise, the small x is used.

PRINT NUMBERS WITHOUT THE TRIPLE ASTERISK: Use VIEW X rather than PRX.

MACHINE STATUS WITH "PRFLAGS": The first five printed lines give the SIZE, the first register of the statistics block, the trigonometric mode, and the display mode. R/S at this point to prevent printing of the flags.

22-4 PRINT LASTX REGISTER ("PRL"): This routine prints the contents of the L Register. Use in conjunction with the printer function "PRSTK" for stack analysis.

LBL "PRL", "L= ", ARCL L, PRA, RTN (16 bytes)

22-5 YELLOW FILTER FOR PHOTOCOPYING: Photocopies of printer tape may be improved with the use of a yellow filter. Try a yellow transparent report cover. Source: Frits Kuyt (236) (PPC CJ, V7N3P5).

22-6 SAVE PAPER WHILE PRINTING BYTES: When finding the number of bytes in a program using the printer and CAT 1, save paper by cataloging in MAN Mode (SST if neccessary) to the step before the END of the program whose byte count is desired, then stop, switch printer to TRACE Mode, and SST to print the bytes in that program.

22-7 PRINT BUFFER: The print buffer is a portion of memory in the printer which holds accumulated characters and columns of dots until the command to print is given. It has a certain number of 'positions' or 'registers', and when they are all filled, the contents are automatically printed (or the first line is automatically printed, if there is more than one line). Each character accumulated into the buffer takes up one position, whether it was generated by ACX, ACA, or ACCHR, and each special dot column takes up one position, whether it was generated by ACCOL or ACSPEC. The SKPCHR command also occupies one position, regardless of the number of characters skipped. SKPCOL uses only one position if the number of columns skipped is a multiple of 7 or is less than 7, and two positions otherwise. The maximum number of characters or columns that can be accumulated under any circumstances is 43, but the buffer often fills up before that number is reached. Why?

Each 'mode change' also takes up one print-buffer position, where a mode is identified by (a) whether the printing is single or double width (determined by Flag 12 at the time the character or column or skip is accumulated), (b) whether a character or a dot column is to be printed, and (c) whether all normally upper-case letters are to be printed as upper or lower case (determined by Flag 13 at the time the character is accumulated). Thus there are 2 cubed or eight of these buffer 'modes' in all (nothing to do with the MAN/TRACE/NORM modes).

When any operation involving an input to the print buffer is performed, the printer looks to see if the new mode (as defined by the new operation along with the calculator's Flags 12 and 13) agree with the old mode (as defined by the last setting). If the new mode agrees with the old mode, then the characters or columns get accumulated with no additional buffer positions taken to indicate the mode. If the modes do not agree, one buffer position is used for a command to shift to the new mode, and then the characters or columns are accumulated. The operations ACA, ACX, ACCHR, and SKPCHR define character mode, while ACCOL, ACSPEC, and SKPCOL define column mode. Other printing operations such as PRX, PRA, program and flag listings, etc., define character mode in that they leave the printer in that mode afterward (they also cause the buffer to be printed out first, if anything has been accumulated into it since the last time it was printed out by other than an overflow). Setting and clearing Flags 12 and 13 do not have any effect on the print buffer unless and until one of the above operations is executed, at which time the modes are compared as described above.

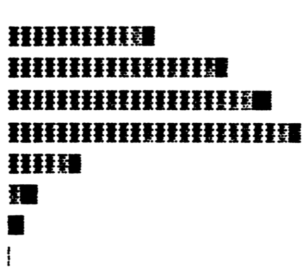
Note that it only takes one position to go from any mode to any other (e.g., from single-width character upper-case to double-width character lower-case). On the other hand, this position may be used even when the mode change turns out to be irrelevant, such as when executing ACA with nothing in the Alpha Register, or changing to lower case mode when the characters accumulated are all non-alphabetic. Operations which only print out what is already in the buffer, namely PRBUF and ADV, do not make use of Flags 12 or 13, either in the printing or in the setting of the printer mode afterward.

As an example of use of the knowledge of print-buffer mode transitions, suppose you want to insert a space between two special characters created using BLDSPEC. Using 7, SKPCOL takes up only one buffer position, since the printer was already in column mode after the first character was ACSPEC'd. But using either 1, SKPCHR; 32, ACCHR; or ACA with a space in Alpha, takes three positions because of the transitions between column and character mode that occur before and after the space.

As one more example of print-buffer usage, consider the printing of lower case. One can save many bytes of program (not to mention execution time) by entering lower-case characters directly in text lines and printing everything with Flag 13 clear, rather than by using ordinary upper-case characters and accumulating them with Flag 13 set to make them lower case. Not only does the former method save program bytes, but it saves print-buffer positions as well. Each time Flag 13 is changed and more characters are accumulated, an extra buffer position is used to make the transition, while no such transition is needed or made if the characters can be accumulated without any flag changes. Using ACCHR with the ASCII (ACCHR) codes for lower case (97-122) also saves on buffer space, but usually not on program bytes.

Source: Roger Hill (4940) (PPC CJ, V7N6P19-20).

22-8 HIGH-RESOLUTION HISTOGRAM ("HG"): Have YMIN in R06, YMAX-YMIN in R07, and plot value in X, then XEQ "HG". The example below is for YMIN = 0, YMAX-YMIN = 100, and x-values of 50, 75, 90, 100, 25, 10, 5, and then 1. Source: Ronald Gordon (3449) (PPC CJ, V7N9P17).



```
01 LBL "HG"   09 +        17 GTO 06   25 31       33 *
02 RCL 06     10 INT      18 LBL 05   26 LBL 01   34 .5
03 -          11 X=0?     19 -        27 ACCHR    35 +
04 RCL 07     12 RTN      20 7        28 DSE Y    36 7
05 /          13 7        21 /        29 GTO 01   37 MOD
06 167        14 X<=Y?    22 INT      30 LBL 02   38 INT
07 *          15 GTO 05   23 X<=0?    31 LASTX
08 1          16 CLX      24 GTO 02   32 7
```

[continued]

```
39 7
40 +
41 LBL 06
42 127
43 LBL 03
44 ACCOL
45 DSE Y
46 GTO 03
47 PRBUF
48 END
```

(71 bytes)

22-9 SYNTHETIC FUNCTION "eGØBEEP" FOR PRINTER FUNCTIONS: "eGØBEEP" is a special synthetic function that can be used to enter printer functions into programs in fewer keystrokes, whether the printer is plugged in or not. It can also be used to execute printer functions in normal or USER Modes.

eGØBEEP #	XROM	FUNCTION
65	29,01	ACA
66	29,02	ACCHR
67	29,03	ACCOL
68	29,04	ACSPEC
69	29,05	ACX
70	29,06	BLDSPEC
71	29,07	LIST
72	29,08	PRA
73	29,09	"PRAXIS"
74	29,10	PRBUF
75	29,11	PRFLAGS
76	29,12	PRKEYS
77	29,13	PRP
78	29,14	"PRPLOT"
79	29,15	"PRPLOTP"
80	29,16	PRREG
81	29,17	PRREGX
82	29,18	PRΣ
83	29,19	PRSTK
84	29,20	PRX
85	29,21	REGPLOT
86	29,22	SKPCHR
87	29,23	SKPCOL
88	29,24	STKPLOT

Assign "eGØBEEP" to a key using a Key Assignments Program (such as "KA" in 'Synthetic Programming on the HP-41C', pp 44-47, 86-87, by William Wickes). The input to the program is 0, ENTER, 167, ENTER, 'nn' (where nn is the keycode of the desired key).

When programming with or without a printer plugged in, when you want a printer function placed into the program, just execute "eGØBEEP" by pressing its assigned key in USER Mode, then supply it with a 2-digit number from 65 to 88, corresponding to the printer functions shown at left. CAUTION!! '89' crashes the HP-41 in Run Mode, and briefly blanks the display in PRGM Mode! In PRGM Mode the code goes in the line as the normal mnemonic if the printer is plugged in, but as its XROM equivalent if it is not. When a printer is subsequently plugged in, the function appears and executes properly.

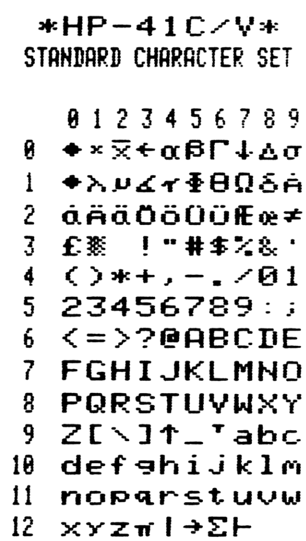
When eGØBEEP is executed when not in PRGM Mode, and then one of the valid eGØBEEP numbers (65 - 88) is supplied, the corresponding printer function is executed. There are TWO EXCEPTIONS: 77 (for PRP) does not work; and 71 (LIST) works differently than normal: rather than prompt for the number of lines to list, it simply begins printing program steps at the current location of the pointer in memory; furthermore, it will stop printing after 71 lines (or after printing an END). If reexecuted, eGØBEEP will cause the printer to advance one line, then print another 71 lines of program.

Source: Robert Edelen (339) (PPC CJ, V7N3P16).

22-10 STANDARD CHARACTER SET ("CE"): This routine will print the 82143A Printer standard character set in a compact matrix, with characters indexed by their "ACCHR" number. To use, just XEQ "CE". Source: Ronald Gordon (3449) (PPC CJ, V7N1 P23) & HP KEY NOTES, V4N2P11.

```
01 LBL "CE"
02 ADV
03 FIX 0
04 CF 29
05 SF 25
06 SF 12
07 CF 13
08 " *HP-41C/V*"
09 PRA
10 CF 12
11 " STANDARD CH"
12 "├ARACTER SET"
13 PRA
14 ADV
15 "0 1 2 3 4 "
16 "├5 6 7 8 9"
17 ACA
18 3
19 SKPCOL
20 .127
21 LBL 00
22 CLA
23 ADV
24 CF 12
25 .009
26 X<>Y
27 STO Z
28 10
29 /
30 INT
31 ARCL X
32 ACA
33 RDN
34 X<>Y
35 SF 12
36 " "
37 ACA
38 LBL 01
39 ACCHR
40 FC? 25
41 GTO 02
42 ISG X
43 ISG Y
44 GTO 01
45 GTO 00
46 LBL 02
47 PRBUF
48 FIX 2
49 SF 29
50 CF 12
51 CLX
52 END
```

(149 bytes)



22-11 DATA NAMES ("DN"): When documenting programs, a listing of the names of data (like 'pointer') in data registers, rather than specific numbers, which may vary, is often useful. Using the printer to generate this list would be handy, especially as it will print many characters not found on a standard typewriter. The routine below makes this convenient. It turns Alpha Mode on and prompts for data name with the register number: input alpha characters only, then R/S. Source: John Dearing (PPC CJ, V8N2P17).

```
01 LBL "DN"             11 "LAST REG. NO.?"  21 "├0"       31 X<=Y?
02 FIX 0                12 PROMPT            22 RDN        32 GTO 01
03 CF 29                13 X<Y?              23 ARCL X     33 FIX 2
04 ADV                  14 GTO 00            24 "├= "      34 SF 29
05 LBL 00               15 X<>Y              25 ACA        35 AOFF
06 "1 ST REG. NO.?"     16 AON               26 PROMPT     36 END
07 PROMPT               17 LBL 01            27 ACA
08 ENTER                18 "R"               28 PRBUF
09 X<0?                 19 10                29 1
10 GTO 00               20 X>Y?              30 +          (89 bytes)
```

EXAMPLES OF USE:

```
R00= POINTER

R09= % DISCOUNT
R10= 1.20945
R11= <USED>

R99= a
R100= b
R101= c
```

22-12 TEXT ("TX"): To print text, XEQ "TX". When "TEXT?" appears, key in up to 24 characters. After the 24th character, hear a tone: press backarrow key to clear the last character(s) if in the middle of a syllable, and key in a hyphen, if desired. Then press R/S. To terminate the text write operation, key in a space (" ") and R/S. You can change step 12 (GTO 00) to GTO "TX", and delete step 05 (LBL 00). NOTE: This same text writing can be done without a routine: just turn on Alpha Mode and key in the line of text as above; when the line is ready to be printed, press the PRINT key on the printer (= PRA) to print the line; key in the next line and repeat. Source: HP KEY NOTES, V4N3P10.

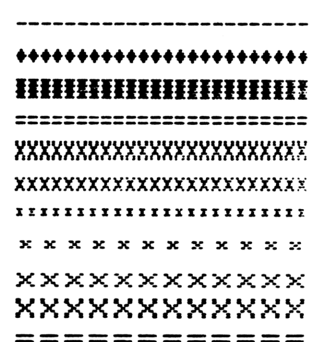
```
01 LBL "TX"     04 AON        07 PROMPT    10 PRBUF     13 AOFF
02 " "          05 LBL 00     08 ACA       11 X≠Y?      14 CLX
03 ASTO X       06 "TEXT?"    09 ASTO Y    12 GTO 00    15 END    (33 bytes)
```

22-13 DIVIDING LINE ("LINE"): This routine uses the hyphen (minus sign) to form a line; with Flag 12 set or clear, it will print a horizontal line. Other symbols can be substituted.

```
LBL "LINE", "------------", FC? 12, "├------------", PRA, RTN      (40 bytes)
```

22-14 PRINTOUT DIVIDERS ("DIV" & "DV"): These routines use no numeric data registers. Execute either; when "ACCHR NO.?" appears, input the ACCHR character number (1-127) (see Printer Handbook, page 37), then R/S. A full line of characters will be printed, whether Flag 12 is set or clear. If R/S is pressed without keying in a number, a full line of dashes (character # 45) will be printed by default. "DIV" uses 1 numeric label and 40 bytes; "DV" uses the Alpha Register but no numeric labels: it is 12 more bytes than "DIV" but much faster. To convert either routine to a subroutinable version with no prompts and no default value, delete "ACCHR NO.?", 45, PROMPT. Have any ACCHR character number in X, then execute the routine. Source: John Dearing (2791) (PPC CJ, V8N2P16).

```
01 LBL "DIV"         10 ACCHR             04 ENTER      13 ASTO X
02 "ACCHR NO.?"      11 DSE Y             05 PROMPT     14 ARCL X
03 45                12 GTO 14            06 BLDSPEC    15 FC? 12
04 PROMPT            13 PRBUF             07 CLA        16 ARCL X
05 12                14 END               08 ARCL X     17 FC? 12
06 FC? 12                                 09 ARCL X     18 ARCL X
07 ST+ X             01 LBL "DV"          10 ARCL X     19 PRA
08 X<>Y              02 "ACCHR NO.?"      11 ASTO X     20 CLX
09 LBL 14            03 45                12 ARCL X     21 END
```



22-15 PRINTER COLUMN ALIGNMENT ("AN" & "P2"): This routine prints one or two columns of numbers with aligned decimal points, by determining the number of print positions to skip before printing each number. After keying in the routine, you may wish to assign "AN" to 11(Σ+) and "P2" to 15 (LN). Instructions:

1. Press A to accumulate the number in X into the print buffer, with Flag 29 clear (no digit grouping) and FIX 2 display; use with numbers of up to 7 digits to the left of the decimal. If the buffer is initially empty, the number (plus spaces for missing leading digits) will be left-justified when the buffer is printed; a single space is also put into the buffer after the number.

2. Optional: to add an Alpha string of up to 12 characters into the buffer, so the number (accumulated in step 1) and the Alpha string will print on the same line, switch to ALPHA Mode, key in string, turn ALPHA off, then press B.

3. Press C to print the buffer. See examples 1 & 2.

4. Repeat steps 1 - 3 as often as desired; successive numbers printed will be aligned on their decimal points.

5. Press D at any time to clear the Alpha Register.

6. To print two numbers on the same line, key in the first number, ENTER, second number; press E to execute "P2". See example 3.

7. To clear the buffer without printing, turn the printer off, then on again.

8. Change steps 04 (FIX 2) and 14 (6) as desired. Example 4 was created with a '3' in step 14. In general, the number in step 14 should be one less than the number of digits needed to the left of the decimal.

```
Ex. 1:        123.45                    Ex. 3:  -9999.99 ABCDEFGHIJKLMNO
          -1234567.12                               5.00 SERVICE CHARGES
Ex. 2:        198.00 KILOMETERS                   400.00 RENT
```

A	B	C	D	E
ACCUMULATE NUMBER	ACCUMULATE ALPHA	PRINT BUFFER	CLEAR ALPHA	PRINT TWO NUMBERS

```
01 LBL "AN"   08 INT      15 -        22 CLX      29 RTN        36 X<>Y
02 LBL A      09 X=0?     16 SKPCHR   23 RTN      30 LBL D      37 XEQ A
03 SF 27      10 ISG X    17 X<>Y     24 LBL B    31 CLA        38 RDN
04 FIX 2      11 ABS      18 ACX      25 ACA      32 RTN        39 RDN
05 CF 29      12 LOG      19 " "      26 RTN      33 LBL "P2"   40 XEQ A
06 CLA        13 INT      20 ACA      27 LBL C    34 LBL E      41 PRBUF
07 ENTER      14 6        21 SF 29    28 PRBUF    35 CF 12      42 END     (76 bytes)
```

Source: Richard Nelson (1) (PPC J, V6N5P31).

22-16 PRINT ALPHA LEFT, X RIGHT ("AX"): To print the contents of the Alpha register left-justified and the contents of the X Register right-justified on the same line, key in control number, S, then XEQ "AX", where the control number = S = the number of characters in Alpha + DSP (the number of decimal places displayed in FIX Mode). For instance, with "NO. 01" in Alpha, 256.98 in X, and FIX 2 Mode, key in '8' and XEQ "AX" to get the first line of the example, below left. ("NO. 01" is 6 characters; DSP = 2; hence S = 6 + 2 = 8.) TIP: to change "NO. 01" to "NO. 02" in the Alpha Register, rather than keying in "NO. 02", you can just press SHIFT, APPEND, BACKARROW, SHIFT, 2.

```
NO. 01          256.98  (S=8)        APPLES               99  (S=5)
NO. 02       58,966.01  ( " )        ORANGES             234  (S=6)
NO. 03       -5,987,63  ( " )        PINEAPPLES            5  (X=9)
```

From S, you may subtract 1 if the decimal is not to be printed (FIX 0, CF 29). If you want the right (numeric) column moved to the left, add the number of spaces you want on the right to S. In the example below (in FIX 2, SF 29 Mode), line 1 has a control number of S+10 = 5+10 = 15 [where S = 5 = no. of Alpha char. (5) + DSP (0)]; line 2 has a control no. of 5+5 = 10; and line 3, a control no. of 5+0 = 5.

```
LARGE     13.               (S = 15)
SMALL           27.         (S = 10)
TOTAL                 42.   (S =  5)
```

If you want to use the same control number where the Alpha strings will be of different length, key in spaces after all Alpha strings but the longest, to make them as long as the longest. For "APPLES", "ORANGES", & "PINEAPPLES", for example, key in 4 spaces after "APPLES" and 3 spaces after "ORANGES" to make them the same length as "PINEAPPLES". NOTE: the LOG function rounds up the logarithms of some large numbers; for these unusual cases, the number will unavoidably be printed displaced one space to the left. Source: William Cheeseman (4381) (PPC CJ, V7N5P8, V7N9P17).

```
01 LBL "AX"
02 21
03 X<>Y
04 -
05 RCL Y
06 ABS
07 X≠0?
08 LOG
09 INT
10 FC? 29
11 GTO 00
12 .75
13 /
14 INT
15 LBL 00
16 -
17 ACA
18 SKPCHR
19 RDN
20 ACX
21 PRBUF
22 END
```
(39 bytes)

22-17 PRINT FUNCTION VALUES ("FN"): This routine prints a table of x- and y-values for any function which returns f(x) for x. The function may then be graphed with PRPLOT without change. To use: 1. Key in your function as a program, with an Alpha label of 6 or fewer characters. Upon entry of your function by either this routine or by PRPLOT, the value in the X Register will also be available in R06. 2. XEQ "FN"; input as prompted: NAME (of your function), Xmin, Xmax, and Xinc (x increment--positive values only). 3. A table of x- and y-values will be printed, with aligned decimal points. 4. If you wish, XEQ "PRPLOT"; use the table just generated to help you select maximum and minimum values of x and y for your plot.

This routine allows printing numbers in two equal columns, with up to nine digits in each number. As it is written, two of these digits will be after the decimal (FIX 2) and up to 7 before it. See first example, below right ("TEST"). If a number is negative, the '-' sign will be printed; if positive, no sign will be printed; the numbers will still line up on the decimals. You can change this format: in general, Step 57 (presently a '6') should be one less than the number of digits to the left of the decimal. Change Step 02 (FIX 2) as needed. The second example below right ("TRIAL") was printed after changing Step 57 to '4' and Step 02 to 'FIX 4'. Source: John Dearing (2791).

Sample Run:

```
 VALUES OF SAMPLE
   X        Y
-------------
  -2.00     2.50
  -1.00     1.50
   0.00     0.00
   1.00    -0.50
   2.00     1.50
   3.00     4.17
   4.00     7.75
```

Sample Function:

```
01♦LBL "SAMPLE"
02 SF 25
03 X↑2
04 LASTX
05 X↑2
06 2
07 /
08 -
09 RCL 06
10 1/X
11 -
12 END
```

More Examples:

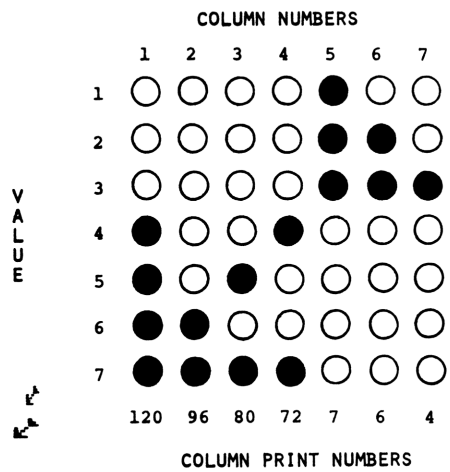
```
   VALUES OF TEST
     X          Y
-------------
1234567.12 -1234567.12

   VALUES OF TRIAL
     X          Y
-------------
12345.1234 -12345.1234
```

```
01 LBL "FN"
02 FIX 2
03 AON
04 "NAME ?"
05 PROMPT
06 AOFF
07 ASTO 11
08 LBL 14
09 "X MIN ?"
10 PROMPT
11 STO 06
12 "X MAX ?"
13 PROMPT
14 STO 09
15 X<Y?
16 GTO 14
17 "X INC ?"
18 PROMPT
19 STO 10
20 ADV
21 "      VALUES OF "
22 ARCL 11
23 CF 12
24 PRA
```

[continued]

25 " X Y"	34 STO 00	43 RCL 06	52 X=0?	61 ACX
26 SF 12	35 RCL 06	44 X<=Y?	53 ISG X	62 " "
27 PRA	36 XEQ 12	45 GTO 13	54 ABS	63 ACA
28 "------------"	37 RCL 00	46 RTN	55 LOG	64 SF 29
29 PRA	38 XEQ 12	47 LBL 12	56 INT	65 END
30 CF 12	39 PRBUF	48 CF 29	57 6	
31 LBL 13	40 RCL 10	49 CLA	58 -	
32 RCL 06	41 ST+ 06	50 ENTER	59 SKPCHR	
33 XEQ IND 11	42 RCL 09	51 INT	60 X<>Y	(161 bytes)

22-18 SYNTHETIC BLDSPEC: Write out a 56-bit binary number; the first 7 bits are always 0001000; the remaining 49 bits are from the 7 x 7 grid of the special character, using 1's for dark dots and 0's for blanks. Start with the lower left corner of the grid and work up the column; then bottom-to-top of each succeeding column. [If the character is in a 7 x 5 matrix (first and last columns of a 7 x 7 matrix blank for spacing), include the bytes for these two columns (all 0's in this case) anyway.]



Group the 56 bits into 4-bit groups, then make a 7-character text line from the hexadecimal equivalents. For the example at right, we have:

0001 0001	1110 0011	0000 0101	0000 1001	0000 0001	1100 0011	0000 0100
1 1	E 3	0 5	0 9	0 1	C 3	0 4

So the text line, preceeded by a Text 7 byte, is hex F7 11 E3 05 09 01 C3 04. The decimal equivalent is: 247, 17, 227, 5, 9, 1, 195, 4.

Key the text line into a program, then follow it with 'RCL M, ACSPEC'. Check to be sure Flag 21 is set before executing the program.

Comparison of byte counts: normal method below is 30 bytes, synthetic method is 12:

Normal: 0, ENTER, 120, BLDSPEC, 96, BLDSPEC, 80, BLDSPEC, 72, BLDSPEC, 7, BLDSPEC, 6, BLDSPEC, 4, BLDSPEC, ACSPEC.

Synthetic: "▩▩天▩天▩工", RCL M, ACSPEC.

Both methods above build the special character and accumulate it into the buffer; to print it, execute PRBUF or ADV. When the synthetic text string above is printed with PRP or LIST, it appears as "Ωβσ×α"; only five characters show because the printer listing of a text line will only show characters from the top half of the Byte Table; characters corresponding to bytes in the lower half of the table are invisible. Furthermore, the print buffer uses bytes from Rows A, B, D & E for internal purposes related to special character printouts, single and double width instructions, etc. Hence, text lines containing characters from those few rows may print out in very strange ways. For example, if a text line contains the character corresponding to byte 'D5', a program listing containing that line will have all printout following that character printed double-wide and lower-case. Source: William Wickes (3735) ('Synthetic Programming on the HP-41C', p 68-70). See 25-5.

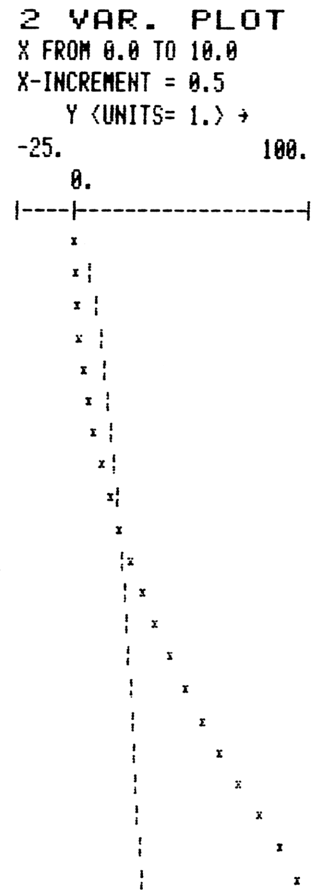
22-19 SYNTHETIC PPC LOGO ("LG"): This routine puts the PPC logo into the printer buffer; print it with PRBUF or ADV. NOTE: Lines 02 & 03 are nonstandard. Line 02 is decimal 254, 17, 194, 228, 124, 60, 122, 241, 17, 102, 62, 30, 61, 120, 249. Line 03 is decimal 248, 127, 17, 158, 29, 155, 191, 78, 135. Be sure Flag 21 is set before attempting to print. Source: Richard Nelson (1) (PPC ROM).

01 LBL "LG" *02 "Ω|<zΩf >£=x" *03 "⊢Ω≠ N" 04 X<> O [continued]

```
05 ACSPEC
06 X<> N
07 ACSPEC
08 X<> M
09 ACSPEC
10 X<> O
11 RTN
```

(45 bytes)

22-20 TWO-VARIABLE PLOTTING ("2V"): In the plotting functions of the printer ROM, only one variable may be plotted. However, an axis character is printed as well, in a column specified by the user. This routine computes and plots this column designation as a point of a separate function, since each REGPLOT or STKPLOT execution may move the axis character to any of the 168 columns. No axis is printed. The routine prompts for "F-1" and "F-2" (the names of the first and second functions), "YMIN" and "YMAX" (minimum and maximum values of Y for both functions), "XMIN" and "XMAX", and also "XINC" (x increment). NOTE: if the value of the second function drops below YMIN, its graph will be distorted—it will be plotted at YMAX until the value rises above YMIN. Upon entering either function, the value of X is also available in Register 06. Zero will be marked on the y-axis only if it is between YMIN and YMAX, inclusive; otherwise, it marks YMIN.



INSTRUCTIONS: Key in the two functions to be plotted as programs, each beginning with a global label of 6 or fewer characters. XEQ "2V"; input as prompted and press R/S. A heading, the y-axis, and the double plot will be printed. Minimum SIZE: 012. EXAMPLE: Key the following functions into program memory: LBL "AA", X↑2, RTN; LBL "BB", SQRT, 10, *, RTN. Next, XEQ "2V" and input as prompted: F-1 = "AA"; F-2 = "BB"; YMIN = '-25'; YMAX = '100'; XMIN = '0'; XMAX = '10'; XINC = '.5'; R/S. The plot shown results.

Source: Jake Schwartz (1820) (PPC CJ, V7N1P24).

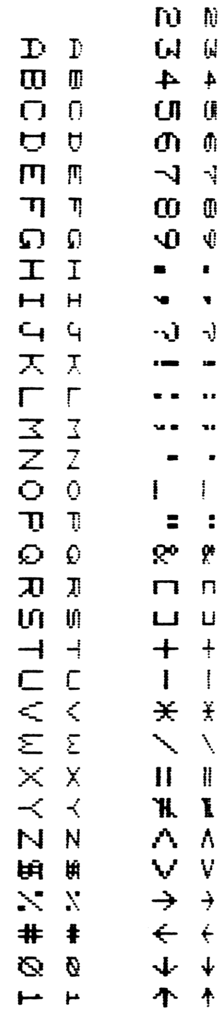
```
01 LBL "2V"
02 AON
03 "F-1?"
04 PROMPT
05 ASTO 07
06 "F-2?"
07 PROMPT
08 AOFF
09 ASTO 08
10 "YMIN?"
11 PROMPT
12 STO 00
13 "YMAX?"
14 PROMPT
15 STO 01
16 "XMIN?"
17 PROMPT
18 STO 05
19 STO 11
20 "XMAX?"
21 PROMPT
22 STO 10
23 "XINC?"
24 PROMPT
25 STO 09
26 RCL 00
27 X>0?
28 GTO 00
29 RCL 01
30 X<0?
31 GTO 00
32 0
33 GTO 02
34 LBL 00
35 RCL 00
36 LBL 02
37 STO 04
38 168
39 STO 02
40 ADV
41 SF 12
42 "2 VAR. PLOT"
43 PRA
44 CF 12
45 FIX 1
46 "X FROM "
47 ARCL 05
48 "├ TO "
49 ARCL 10
50 PRA
51 "X-INCREMENT = "
52 ARCL 09
53 PRA
54 XROM "PRAXIS"
55 RCL 11
56 STO 06
57 LBL 01
58 XEQ IND 08
59 RCL 00
60 -
61 RCL 01
62 RCL 00
63 -
64 /
65 168
66 *
67 R↑
68 CLX
69 X=Y?
70 1
71 +
72 INT
73 1 E3
74 /
75 168
76 +
77 STO 02
78 RCL 06
79 XEQ IND 07
80 REGPLOT
81 RCL 09
82 ST+ 06
83 RCL 10
84 RCL 06
85 X<=Y?
86 GTO 01
87 END
```

(192 bytes)

22-21 SIDEWAYS PRINTER CHARACTERS ("PRSW"): Use these BLDSPEC instruction codes to build a character set which will print longwise on the paper. Store these in the data registers, and recall them and ACSPEC them into the print buffer as needed. Before accumulating them, CF 12 for single-height characters or SF 12 for double-height characters. Due to the 44-position buffer restriction, only 5 'rows' can be positioned on the printer paper at once. It is suggested that each row be separated by 1 - 7 columns, using SKPCOL. Example:

```
RCL 01, ACSPEC, 3, SKPCOL   (8 buffer positions)
RCL 02, ACSPEC, 3, SKPCOL   (         "         )
RCL 03, ACSPEC, 3, SKPCOL   (         "         )
RCL 04, ACSPEC, 3, SKPCOL   (         "         )
RCL 05, ACSPEC, PRBUF       (7 buffer positions)
```

Output of this example: 



Here is the routine used to print the full set of 64 characters, both regular height and double height, as shown at the right:

```
01 LBL "PRSW"
02 ADV
03 .031
04 LBL 00
05 XEQ 01
06 6
07 SKPCHR
08 RCL Z
09 32
10 +
11 XEQ 01
12 PRBUF
13 X<>Y
14 32
15 -
16 ISG X
17 GTO 00
18 RTN
19 LBL 01
20 RCL IND X
21 SF 12
22 ACSPEC
23 1
24 SKPCHR
25 RDN
26 CF 12
27 ACSPEC
28 END
```

Source: Richard Nelson (1) (PPC CJ, V7N1P23).

Registers, characters, and BLDSPEC codes for building the characters:

R00	space	0	0	0	0	0	0	0
R01	A	65	65	127	65	65	34	28
R02	B	63	65	65	63	65	65	63
R03	C	62	65	1	1	1	65	62
R04	D	31	34	66	66	66	34	31
R05	E	127	1	1	31	1	1	127
R06	F	1	1	1	31	1	1	127
R07	G	94	97	113	1	1	65	62
R08	H	65	65	65	127	65	65	65
R09	I	62	8	8	8	8	8	62
R10	J	14	17	17	16	16	16	124
R11	K	65	33	17	15	17	33	65
R12	L	127	1	1	1	1	1	1
R13	M	65	65	65	73	85	99	65
R14	N	65	97	81	73	69	67	65
R15	O	28	34	65	65	65	34	28
R16	P	1	1	1	63	65	65	63
R17	Q	92	34	81	65	65	34	28
R18	R	65	33	17	63	65	65	63
R19	S	63	64	64	62	1	1	126
R20	T	8	8	8	8	8	8	127
R21	U	62	65	65	65	65	65	65
R22	V	8	20	20	34	34	65	65
R23	W	34	85	73	73	65	65	65
R24	X	65	34	20	8	20	34	65
R25	Y	8	8	8	8	20	34	65
R26	Z	127	2	4	8	16	32	127
R27	$	63	84	84	62	21	21	126
R28	%	97	98	4	8	16	35	67
R29	#	20	20	127	20	127	20	20

R30	0	29	34	69	73	81	34	92
R31	1	62	8	8	8	8	12	8
R32	2	127	1	2	60	64	65	62
R33	3	62	65	64	56	16	32	127
R34	4	16	16	127	18	20	24	16
R35	5	62	65	64	64	63	1	127
R36	6	60	66	65	63	1	2	124
R37	7	4	4	4	8	16	33	127
R38	8	62	65	65	62	65	65	62
R39	9	12	16	32	126	65	65	62
R40	.	28	28	28	0	0	0	0
R41	,	4	8	28	28	0	0	0
R42	?	8	0	8	48	64	65	62
R43	!	12	0	12	12	12	12	12
R44	:	12	12	0	0	0	12	12
R45	;	4	8	12	0	0	12	12
R46	'	0	0	0	0	12	12	12
R47	_	127	0	0	0	0	0	0
R48	"	0	0	0	0	54	54	54
R49	&	110	17	41	70	6	9	6

[continued]

R50	[	62	2	2	2	2	2	62	R57	≠	1	2	127	8	127	32	64
R51	]	62	32	32	32	32	32	62	R58	<	96	24	6	1	6	24	96
R52	+	8	8	8	127	8	8	8	R59	>	3	12	48	64	48	12	3
R53	-	0	0	0	127	0	0	0	R60	↑	8	8	8	73	42	28	8
R54	*	8	73	42	29	42	73	8	R61	↓	8	28	42	73	8	8	8
R55	/	1	2	4	8	16	32	64	R62	→	8	16	32	127	32	16	8
R56	=	0	127	0	0	127	0	0	R63	←	8	4	2	127	2	4	8

22-22 VERTICAL ACCUMULATION OF 2-DIGIT NUMBERS ("V2"): This routine accumulates into the print buffer a small 2-digit integer rotated 90° clockwise from normal. It is useful when printing indentifier for histogram bars or plotting. The characters are build on a 3x5 matrix, and thus the 2 digits occupy only 5 print columns. Taller characters can be generated by setting Flag 12. Spacing on either side is not provided, in order that these characters can be placed right up to either margin of the print line. Spacing can be provided by the user with SKPCOL. This routine uses no data registers; it uses the stack. To use, key in any 2-digit integer, XEQ "V2". Source: Cliff Carrie (834) (PPC CJ, V8N1P15).

```
01 LBL "V2"
02 10
03 /
04 ENTER
05 FRC
06 10
07 *
08 XEQ IND Y
09 XEQ IND Y
10 LBL 14
11 10
12 *
13 FRC
14 LASTX
15 INT
16 16
17 *
18 RCL Z
19 10
20 *
21 +
22 ACCOL
23 FRC
24 X<>Y
25 X≠0?
26 GTO 14
27 RTN
28 LBL 00
29 .25552
30 RTN
31 LBL 01
32 .22232
33 RTN
34 LBL 02
35 .72452
36 RTN
37 LBL 03
38 .34243
39 RTN
40 LBL 04
41 .47564
42 RTN
43 LBL 05
44 .34317
45 RTN
46 LBL 06
47 .25316
48 RTN
49 LBL 07
50 .22247
51 RTN
52 LBL 08
53 .25252
54 RTN
55 LBL 09
56 .34652
57 END
```

(124 bytes)

22-23 PRINTER COMPATIBILITY: This shows how to get various combinations of printing, not printing, stopping, not stopping, pausing, etc, when either a printer is on line (plugged in and on) or no printer is plugged in.

Print OR Stop	1	PRINTER--print without stopping NO PRINTER--stop to view	..., SF 21, "MSG", AVIEW, ...
Print OR Pause	2	PRINTER--print without pausing NO PRINTER--pause to view	..., CF 21, FS? 55, SF 21, "MSG", AVIEW, FC? 55, PSE, ...
Print & Stop	3	PRINTER--print and stop NO PRINTER--stop to view	..., SF 21, "MSG", AVIEW, FS? 55, STOP, ...
Print & Pause	4	PRINTER--print and pause NO PRINTER--pause to view	..., CF 21, FS? 55, SF 21, "MSG", AVIEW, PSE, ...
NO Print & Stop	5	PRINTER--stop and don't print NO PRINTER--stop to view	..., "MSG", PROMPT, ...
NO Print & Pause	6	PRINTER--pause and don't print NO PRINTER--pause to view	..., CF 21, "MSG", AVIEW, PSE, ...
Print OR NO Stop	7	PRINTER--print NO PRINTER--don't pause or stop	..., CF 21, FS? 55, SF 21, "MSG", FS? 55, AVIEW, ...

Example: at the end of an initialization routine, the program is to pause after a first message, then stop after a second message. Both are to print if the printer is on line; subsequent outputs are to print without stopping, or stop if no printer:

..., CF 21, FS? 55, SF 21, "MSG1", AVIEW, PSE, "MSG2", AVIEW, SF 21, STOP, ...

See routines 2-31, 3-12, 4-1, 4-18, 6-4, 6-6, 6-14.

22-24 AUTOMATIC PRINTING OF MULTIPLE OUTPUTS ON ONE OR MORE LINES: When you don't wish to print output that is generated within a loop, line by line, include the following routine within the loop (here, a loop counter is stored in R00):

..., LBL 01, ..., " ", ARCL X, ACA, ..., DSE 00, GTO 01, PRBUF, ...

The print buffer will accumulate 24 alpha characters, and then automatically print before accepting additional characters. To reduce ambiguities when a numerical output is split between two lines, the space (" ") may be replaced by "X". Alternatively, if the output consists of integers, the space can be eliminated if FIX 0, SF 29 Mode is used, in which case the radix will serve to separate successive outputs.

Some examples: 1) Printing successive sums of a series. 2) Printing all the prime numbers within a specified interval. The following routine example shows how the output may be arranged in easy-to-read columns. "=0" prints all the numbers in an interval for which the operations "SQRT, X↑2" return exactly the same number to X. Before execution, "=0" expects the lowest number to be tested in Y, and the highest number in X. The example above right will print with an input of 100, ENTER, 136. The user may insert a line of one or more alpha spaces, followed by "ACA", after line 10 (ADV) to indent the first line or to compensate for initial outputs of different character length from that of subsequent ones. Source: Robert Swanson (5993). See 15-8.

```
100 101 102 106 109 110
114 118 120 121 126 127
128 129 130 135 136
```

01 LBL "=0"	05 -	09 ENTER	13 X↑2	17 ACA	21 DSE Z	25 SF 29
02 FIX 0	06 ISG X	10 ADV	14 " "	18 SIGN	22 GTO 01	26 BEEP
03 CF 29	07 STO X	11 LBL 01	15 ARCL Y	19 +	23 PRBUF	27 END
04 X<>Y	08 LASTX	12 SQRT	16 X=Y?	20 ENTER	24 FIX 2	(46 bytes)

22-25 SYNTHETIC SPECIAL CHARACTERS ("SC", "SCT", "SCL" & "SCX"): This routine allows the user to bring nonstandard print characters, in groups of three, into consecutive registers of main data memory. They then may be ACSPECed and PRBUFed. A complete list of the characters available is shown in the Special Characters Table and the Special Characters List below. The nineteen groups of characters are indicated in the table.

Instructions: Load Registers 06-08 as shown:

R06= # of first group desired, l
R07= # of last group desired, m
R08= # of first register to store characters in (R09 or higher), n

Next, XEQ "SC"; the characters in groups l through m will then be loaded into registers n and above. This routine primarily uses Status Registers M, N and O to do its storing and recalling. The actual characters are brought into Alpha by either 7 or 14 character text lines, which contain sprinklings of buffer control characters, causing the program listing to print oddly. The last group of characters is also returned to the stack (Z, Y & T Registers). Be sure Flag 21 is set before using the special characters.

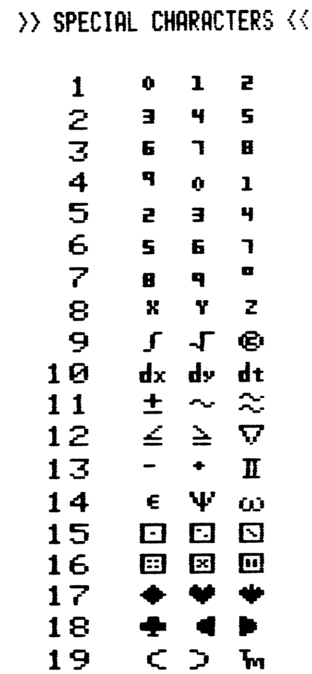
Example 1: Put group 9 (∫, √‾ and ©) in Registers 9, 10 and 11. Solution: key in 9, STO 06, STO 07, STO 08, XEQ "SC" (minimum SIZE 012); the characters will also be returned to the stack:

R09 = Z = ∫
R10 = Y = √‾
R11 = X = ©

Now, to put √‾ in the print buffer (for example), use RDN (or RCL 10), ACSPEC; print with PRBUF or ADV.

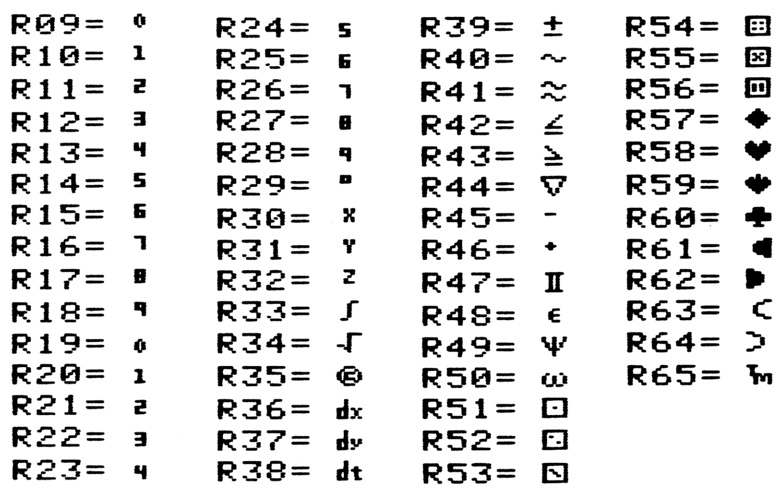
Example 2: Print a table of special characters, as shown below. Key in the "SCT" routine below, then, with SIZE 066 or greater, key in 1, STO 06, 19, STO 07, 9, STO 08, XEQ "SC", XEQ "SCT".

```
01 LBL "SCT"            16 LBL 14      31 RDN
02 ADV                  17 X<Y?        32 PRBUF
03 CF 12                18 XEQ 13      33 ISG X
04 ">> SPECIAL CH"      19 ACX         34 GTO 14
05 "├ARACTERS <<"       20 XEQ 13      35 FIX 2
06 PRA                  21 XEQ 13      36 SF 29
07 ADV                  22 LBL 12      37 CF 12
08 SF 12                23 RCL IND Z   38 RTN
09 FIX 0                24 ACSPEC      39 LBL 13
10 CF 29                25 XEQ 13      40 " "
11 9.011                26 RDN         41 ACA
12 ENTER                27 ISG Z       42 END
13 1.019                28 GTO 12
14 10                   29 .003
15 X<>Y                 30 ST+ T       (115 bytes)
```

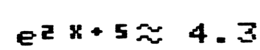


Example 3: Print a list of special characters, as shown below. Key in the "SCL" routine; then, if not previously done, place the special characters in R09-R65 (SIZE 066, 1, STO 06, 19, STO 07, 9, STO 08, XEQ "SC"); finally, XEQ "SCL".

```
01 LBL "SCL"     16 ARCL X
02 ADV           17 "├= "
03 SF 12         18 ACA
04 FIX 0         19 RCL IND X
05 CF 29         20 ACSPEC
06 9.065         21 ADV
07 "R09= "       22 RDN
08 ACA           23 ISG X
09 RCL IND X     24 GTO 11
10 ACSPEC        25 CF 12
11 ADV           26 FIX 2
12 RDN           27 SF 29
13 ISG X         28 END
14 LBL 11
15 "R"           (65 bytes)
```



Example 4: Print the following: $e^{2x+5} \approx 4.3$. With the special characters loaded in R09-R65 as in the examples above, XEQ "SCX".



```
01 LBL "SCX"   05 ACCHR    09 ACSPEC   13 ACSPEC   17 ACX      21 END
02 FIX 1       06 RCL 11   10 RCL 46   14 RCL 41   18 PRBUF
03 SF 12       07 ACSPEC   11 ACSPEC   15 ACSPEC   19 CF 12
04 101         08 RCL 30   12 RCL 14   16 4.3      20 FIX 2    (48 bytes)
```

The characters in the last two groups can be used to print the phases of the moon; accumulate the first two characters of group 19, one after the other, for example, to place a full moon in the buffer. See PPC Calculator Journal, V7N10P14, for more examples.

Source: Jake Schwartz (1820) (PPC CJ, V7N10P11-15).

[continued]

Program Listing:

```
01♦LBL "SC"
02 RCL 07
03  E3
04 /
05 ST+ 06
06 .9
07 ST+ 08
08 GTO IND 06
09♦LBL 01
10 "θ♦qΩ♦♦θ♦♦♦♦"
11 "⊢θ♦R
♦♦"
12 GTO 00
13♦LBL 02
14 "θ♦        S
♦♦θ♦8C
♦♦"
15 "⊢θ♦S♦♦"
16 GTO 00
17♦LBL 03
18 "θ♦S♦♦θ♦ΔÄ
♦♦"
19 "⊢θ♦S
♦♦"
20 GTO 00
21♦LBL 04
22 "θ♦8S
♦♦θ×G♦♦♦"
23 "⊢θx̄'♦♦♦"
24 GTO 00
25♦LBL 05
26 "θ←    K♦♦θx̄     0♦
♦"
27 "⊢θ♦‡♦♦"
28 GTO 00
29♦LBL 06
30 "θx̄N♦♦θ←N♦♦"
31 "⊢θ♦ 0♦♦"
32 GTO 00
33♦LBL 07
34 "θ←0♦♦θ♦0♦♦"
35 "⊢θ♦8P
♦♦"
36 GTO 00
37♦LBL 08
38 "θ♦×
♦θ♦♦3♦"
39 "⊢θ♦×     ♦"
40 GTO 00
41♦LBL 09
42 "θx̄α↓θθθ‡
θ"
43 "⊢θqÄ
Ωe"
44 GTO 00
45♦LBL 10
46 "ΩGδōHΩGāθō"
47 "⊢ΩGx̄>H"
48 GTO 00
49♦LBL 11
50 "θx̄$K"♦θ  ÄαΔΔ"
51 "⊢θäΩ"""
52 GTO 00
53♦LBL 12
54 "θx̄        θx̄♦(♦"
55 "⊢θμkō,F"
56 GTO 00
57♦LBL 13
58 "θ♦ θ♦♦θ♦
♦♦"
59 "⊢θx̄‡?
"
60 GTO 00
61♦LBL 14
62 "θ♦×J♦θμa‡Γ←"
63 "⊢θ$↓θ"8"
64 GTO 00
65♦LBL 15
66 "ΩμOθ'ΩμX4'"
67 "⊢ΩμY4'"
68 GTO 00
69♦LBL 16
70 "Ω∡X5'Ω∡Y5'"
71 "⊢Ω∡7'"
72 GTO 00
73♦LBL 17
74 "θ Δθ8"
75 "⊢θθ‡μ"
76 GTO 00
77♦LBL 18
78 "θpeθ♦× "
79 "⊢Ω♦♦"
80 GTO 00
81♦LBL 19
82 "θ♦×HΩΓ♦$G♦♦"
83 "⊢θα⊢μαx"
84♦LBL 00
85 RCL ]
86 STO IND 08
87 ISG 08
88 RCL \
89 STO IND 08
90 ISG 08
91 RCL [
92 STO IND 08
93 ISG 08
94 ISG 06
95 GTO IND 06
96 END
END        560 BYTES
```

Bytes for creating the synthetic lines above, using "LB": Compare this table with the routine listing, above. The "PROMPT" column is the byte number that should be prompted for by "LB" just before keying in the first byte in a given line. NOTE: The "LB" buffer needs to be at least 476 +'s! Read Chapter XXV, including the examples, before attempting to load these bytes to create the "SC" routine.

LINE NO.	1st "LB" PROMPT	BYTES
03	[1]	27, 19
10	[3]	254, 16, 0, 113, 17, 192, 0, 0, 16, 0, 137, 242, 0, 0, 0
11	[18]	248, 127, 16, 0, 233, 82, 224, 0, 0
14	[27]	254, 16, 0, 169, 83, 224, 0, 0, 16, 0, 56, 67, 224, 0, 0
15	[42]	248, 127, 16, 0, 185, 83, 160, 0, 0
18	[51]	254, 16, 0, 249, 83, 160, 0, 0, 16, 0, 8, 19, 224, 0, 0
19	[66]	248, 127, 16, 0, 249, 83, 224, 0, 0
22	[75]	254, 16, 0, 56, 83, 224, 0, 0, 16, 1, 196, 71, 0, 0, 0
23	[90]	248, 127, 16, 2, 39, 200, 0, 0, 0
26	[99]	254, 16, 3, 165, 75, 128, 0, 0, 16, 2, 165, 79, 128, 0, 0
27	[114]	248, 127, 16, 0, 225, 15, 128, 0, 0
30	[123]	254, 16, 2, 229, 78, 128, 0, 0, 16, 3, 229, 78, 128, 0, 0
31	[138]	248, 127, 16, 0, 32, 79, 128, 0, 0
34	[147]	254, 16, 3, 229, 79, 128, 0, 0, 16, 0, 225, 79, 128, 0, 0
35	[162]	248, 127, 16, 0, 56, 80, 224, 0, 0
38	[171]	254, 16, 0, 1, 176, 134, 192, 0, 16, 0, 0, 51, 128, 192, 0
39	[186]	248, 127, 16, 0, 1, 146, 164, 192, 0

42	[195]	254, 16, 2, 4, 7, 192, 64, 128, 16, 64, 130, 15, 224, 64, 129
43	[210]	248, 127, 16, 113, 21, 218, 181, 81, 28
46	[219]	254, 17, 226, 71, 240, 18, 24, 72, 17, 226, 71, 240, 22, 16, 24
47	[234]	248, 127, 17, 226, 71, 240, 2, 62, 72
50	[243]	254, 16, 2, 36, 75, 241, 34, 0, 16, 32, 32, 65, 4, 8, 8
51	[258]	248, 127, 16, 136, 137, 20, 81, 34, 34
54	[267]	254, 16, 2, 133, 138, 148, 168, 128, 16, 2, 141, 42, 150, 40, 0
55	[282]	248, 127, 16, 12, 107, 24, 44, 70, 131
58	[291]	254, 16, 0, 32, 64, 128, 0, 0, 16, 0, 32, 224, 128, 0, 0
59	[306]	248, 127, 16, 2, 15, 248, 63, 224, 128
62	[315]	254, 16, 0, 1, 197, 74, 128, 0, 16, 12, 97, 15, 228, 6, 3
63	[330]	248, 127, 16, 226, 36, 7, 16, 34, 56
66	[339]	254, 17, 254, 12, 25, 48, 96, 255, 17, 254, 12, 88, 52, 96, 255
67	[354]	248, 127, 17, 254, 12, 89, 62, 96, 255
70	[363]	254, 17, 254, 13, 88, 53, 96, 255, 17, 254, 13, 89, 53, 96, 255
71	[378]	248, 127, 17, 254, 13, 216, 55, 96, 255
74	[387]	254, 16, 32, 227, 239, 239, 142, 8, 16, 56, 251, 239, 143, 143, 142
75	[402]	248, 127, 16, 48, 243, 207, 239, 15, 12
78	[411]	254, 16, 112, 229, 255, 247, 206, 28, 16, 0, 1, 199, 206, 191, 255
79	[426]	248, 127, 17, 255, 251, 231, 199, 0, 0
82	[435]	254, 16, 0, 1, 196, 72, 160, 193, 17, 6, 10, 36, 71, 0, 0
83	[450]	248, 127, 16, 4, 127, 145, 12, 4, 120
85	[459]	144, 119
88	[461]	144, 118
91	[463]	144, 117

CHAPTER XXIII

BANNERS

23-1 BANNER PRINTER ("BANR", "CHAR" & "CODE"): This routine is used to print banners. 'Compression codes' are included for 95 characters. The routine will fit on one track; 7 tracks are need to store all the characters on magnetic cards. The first ten data registers contain the character building blocks; R10 is used by the routine, and R11-R105 contain the codes, which are used to build the corresponding characters. Note that Flag 12 must be set to get full-height characters, or clear for half-height characters; Flag 12 can be set or cleared between characters to print mixed full- and half-height characters.

Instructions: There are three ways to use this routine: (1) XEQ "BANR"; see "CHAR. NO.?"; input the character number (0 = space, 1 = A, etc), press R/S; the character corresponding to that character number will be printed. The prompt will reappear; repeat. (2) You can also key in the character number and XEQ "CHAR"; this label is primarily for use in any program that automatically executes this routine. (3) Key in any of the 'compression codes', including any of your own (for example, key in .1802020218 for 'A'), then XEQ "CODE"; the corresponding character will be printed. If memory is short, just key in the routine and the character building blocks (R00-R09) with a SIZE 011, and use "CODE".

Building the building blocks: In normal or USER Mode, key in '0', press ENTER (to clear X & Y); key in '31', XEQ "BLDSPEC"; Alpha character 31 is now in X (but displays as a boxed star) (other characters can be used). Now use ARCL X and SPACE in ALPHA Mode to build the building blocks one at a time, storing them in the appropriate registers. For example: For R00: CLA, SPACE 6 times, ASTO 00. For R01: CLA, ARCL X 6 times, ASTO 01. For R02: CLA, ARCL X, SPACE 4 times, ARCL X, ASTO 02.

```
01 LBL "BANR"
02 "CHAR. NO.?"
03 PROMPT
04 XEQ "CHAR"
05 GTO "BANR"
06 LBL "CHAR"
07 .004
08 STO 10
09 RDN
10 X=0?
11 GTO 00
12 10
13 +
14 RCL IND X
15 LBL 00
16 CLA
17 10
18 *
19 ARCL IND X
20 FRC
21 10
22 *
23 ARCL IND X
24 FRC
25 PRA
26 ISG 10
27 GTO 00
28 ADV
29 ADV
30 RTN
31 LBL "CODE"
32 .004
33 STO 10
34 RDN
35 GTO 00
36 END
```

(96 bytes)

Example: Here is a routine that will print all 95 characters: LBL "X", 1.095, LBL 01, ENTER, XEQ "CHAR", RCL Z, ISG X, GTO 01, RTN.

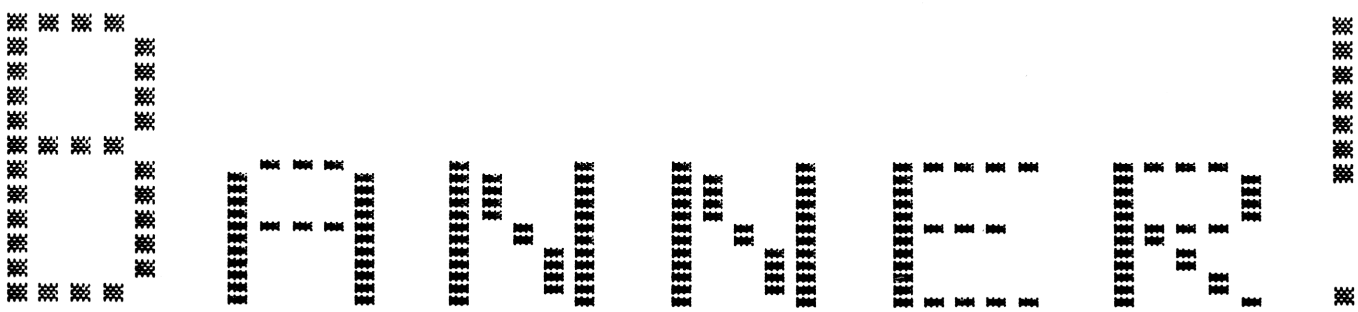
Source: Dean Lampman (41) (PPC J, V6N6P16).

Data Registers, Compression Codes, Character Numbers and Symbols:

```
R00 = "      "
R01 = "██████"
R02 = "█    █"
R03 = "     █"
R04 = "█     "
R05 = " ████ "
R06 = " ██   "
R07 = "   ██ "
R08 = "█████ "
R09 = " █████"
R10 = 0              0 = space
R11 = .1802020218    1 = A
R12 = .1142424295    2 = B
R13 = .9843434367    3 = C
R14 = .1143434398    4 = D
R15 = .1142424243    5 = E
R16 = .1102020203    6 = F
R17 = .9843432353    7 = G
R18 = .1104040411    8 = H
R19 = .4343114343    9 = I
R20 = .5040439103   10 = J
R21 = .1134766743   11 = K
```

[continued]

R22 =	.1140404040	12 = L	R50 =	.6	40 = .	R78 =	.7676117676	68 = ≠
R23 =	.1107060711	13 = M	R51 =	.406	41 = ,	R79 =	.0606060031	69 = $^{-1}$
R24 =	.1105345011	14 = N	R52 =	.0703130205	42 = ?	R80 =	.3476556743	70 = <
R25 =	.9843434398	15 = O	R53 =	.21	43 = !	R81 =	.4367557634	71 = >
R26 =	.1102020205	16 = P	R54 =	.67	44 = :	R82 =	.8024854743	72 = ∡
R27 =	.9843436398	17 = Q	R55 =	.4067	45 = ;	R83 =	.4542114293	73 = $
R28 =	.1132726245	18 = R	R56 =	.000007	46 = '	R84 =	.0418424347	74 = £
R29 =	.0542424293	19 = S	R57 =	.00006	47 = .	R85 =	.9843114367	75 = ¢
R30 =	.0303110303	20 = T	R58 =	.00070007	48 = "	R86 =	.4760980763	76 = %
R31 =	.9140404091	21 = U	R59 =	.5522526580	49 = &	R87 =	.6711671167	77 = #
R32 =	.0190409001	22 = V				R88 =	.9843464265	78 = @
R33 =	.1160706011	23 = W	R60 =	.0000114343	50 = [	R89 =	.05020205	79 = °
R34 =	.8976347689	24 = X	R61 =	.434311	51 =]			
R35 =	.0934803409	25 = Y	R62 =	.0034554343	52 = {	R90 =	.9044449044	80 = α
R36 =	.4383224943	26 = Z	R63 =	.43435534	53 = }	R91 =	.1862426295	81 = β
			R64 =	.0000986743	54 = (	R92 =	.1103030303	82 = Γ
R37 =	.1134343434	27 = ├	R65 =	.436798	55 =)	R93 =	.8028432880	83 = Δ
R38 =	.0003090300	28 = ᵀ	R66 =	.0607110706	56 = ↑	R94 =	.9822222298	84 = θ
R39 =	.3074967430	29 = ◆	R67 =	.7060116070	57 = ↓	R95 =	.1060406090	85 = μ
			R68 =	.2267987634	58 = →	R96 =	.3014041406	86 = π
R40 =	.9823224298	30 = 0	R69 =	.3476986722	59 = ←	R97 =	.4389892222	87 = Σ
R41 =	.4047114040	31 = 1				R98 =	.5020209030	88 = σ
R42 =	.8723424245	32 = 2	R70 =	.3434983434	60 = +	R99 =	.3030103004	89 = τ
R43 =	.6743424295	33 = 3	R71 =	.3434343434	61 = -			
R44 =	.3130301130	34 = 4	R72 =	.00763476	62 = x	R100 =	.1111111111	90 = ▮
R45 =	.6142424293	35 = 5	R73 =	.6776117667	63 = *	R101 =	.0303030303	91 = ‾
R46 =	.9823232357	36 = 6	R74 =	.3434223434	64 = ÷	R102 =	.4040404040	92 = _
R47 =	.0383330209	37 = 7	R75 =	.4060980703	65 = /	R103 =	.05020202	93 = ᶜ
R48 =	.9542424295	38 = 8	R76 =	.0307986040	66 = \	R104 =	.5020202050	94 = o
R49 =	.0502020218	39 = 9	R77 =	.7676767676	67 = =	R105 =	.1802184242	95 = Æ



23-2 LETTER BANNER ("LET", "DIG" & "SYM"): This banner-printing routine, with appropriate 'compression codes' loaded in Registers 01-94, will print the letters, the digits, and 45 other characters. Each banner character printed is composed of the characters themselves. With Flag 12 set, the characters are full-height; with Flag 12 clear, they are half-height (the banner letters will be composed of lower-case letters if Flag 12 is clear). NOTE: this routine is very slow: writing a routine to print your banner is suggested. To print "Banner!", for instance, use the following routine:

```
01 LBL "X"
02 SF 12
03 2
04 XEQ "LET"
05 CF 12
06 1
07 XEQ "LET"
08 14
09 XEQ "LET"
10 14
11 XEQ "LET"
12 5
13 XEQ "LET"
14 18
15 XEQ "LET"
16 SF 12
17 33
18 XEQ "SYM"
19 BEEP
20 OFF
21 END
```
(62 bytes)

Instructions: Load routine and data. Set or clear Flag 12 as desired. To print a letter: enter its character code (1 - 26; see table below), XEQ "LET". To print a digit: enter the digit, XEQ "DIG". To print a symbol: enter its character code, XEQ

"SYM". To 'print' a space, enter '32', XEQ "SYM". The letter banner routine below will fit on 3 tracks; if all the compression codes are stored in data registers, 6 tracks will be needed to save them on magnetic cards. To print letters only, set SIZE 039 before reading the data cards.

CHARACTER CODES & SYMBOLS:

Letters			Digits	Symbols				
1 = A	11 = K	21 = U	0 = 0	3 = ←	29 = ≠	39 = '	58 = :	93 =]
2 = B	12 = L	22 = V	1 = 1	4 = α	30 = £	40 = (	59 = ;	94 = ↑
3 = C	13 = M	23 = W	2 = 2	5 = β	31 = ▮	41 =)	60 = <	95 = _
4 = D	14 = N	24 = X	3 = 3	6 = Γ	32 =	42 = *	61 = =	96 = ⊤
5 = E	15 = O	25 = Y	4 = 4	7 = ↓	33 = !	43 = +	62 = >	
6 = F	16 = P	26 = Z	5 = 5	8 = Δ	34 = "	44 = ,	63 = ?	123 = π
7 = G	17 = Q		6 = 6	9 = σ	35 = #	45 = -	64 = @	124 = |
8 = H	18 = R		7 = 7	10 = ◆	36 = $	46 = .		125 = →
9 = I	19 = S		8 = 8		37 = %	47 = /	91 = [	126 = Σ
10 = J	20 = T		9 = 9		38 = &		92 = \	127 = ⊦

```
LBL "LET"
1
X>Y?
GTO 08
CLX
27
X<=Y?
GTO 08
X<>Y
STO 38
10
+
96
ST+ 38
32
FS? 12
ST- 38
.004
STO 10
RCL IND T
GTO 10
LBL "DIG"
X<0?
GTO 08
10
X<=Y?
GTO 08
X<>Y
48
+
STO 38
8
-
.004
STO 10
RCL IND Y
GTO 10
LBL "SYM"
STO 38
3
X>Y?
GTO 08
CLX
11
X>Y?
GTO 04
CLX
29
X>Y?
GTO 08
CLX
48
X>Y?
GTO 05
CLX
58
X>Y?
GTO 08
CLX
65
X>Y?
GTO 06
CLX
91
X>Y?
GTO 08
CLX
97
X>Y?
GTO 07
CLX
123
X>Y?
GTO 08
CLX
128
X<=Y?
GTO 08
CLX
33
-
GTO 09
LBL 04
CLX
47
+
GTO 09
LBL 05
CLX
29
+
GTO 09
LBL 06
CLX
19
+
GTO 09
LBL 07
CLX
7
-
GTO 09
LBL 08
"BAD CODE"
PROMPT
RTN
LBL 09
.004
STO 10
RCL IND Y
LBL 10
10
*
RCL IND X
XEQ 00
FRC
10
*
RCL IND X
XEQ 00
FRC
PRBUF
ISG 10
GTO 10
ADV
ADV
RTN
LBL 00
.005
STO 37
RDN
LBL 03
10
*
ENTER
INT
X=0?
GTO 01
RCL 38
GTO 02
LBL 01
32
LBL 02
ACCHR
RDN
RDN
FRC
ISG 37
GTO 03
RDN
END
```

(262 bytes)

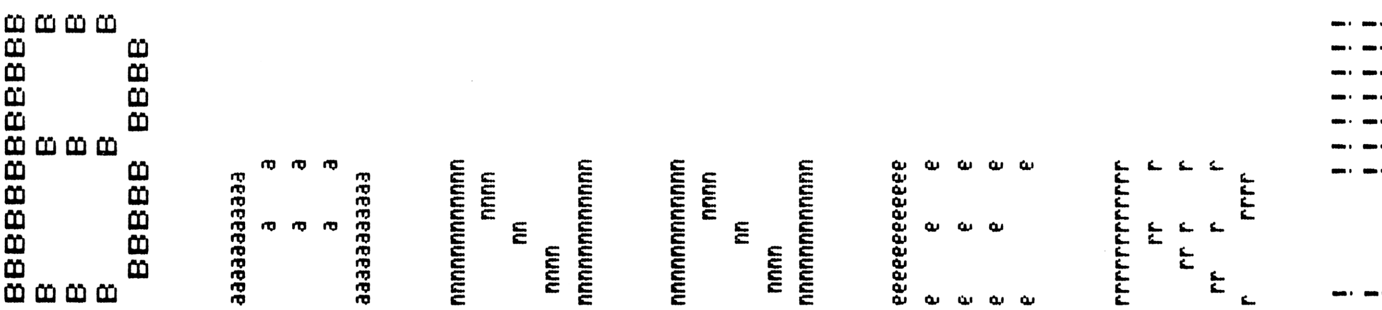
DATA: Below is the data that must be loaded for this routine. R10 is the main routine loop counter, R37 is the subroutine loop counter, and R38 is used by the routine to store the character code. R39 is not used.

R00= 0	R03= 0.000001	R06= 0.011000	R09= 0.011111
R01= 0.111111	R04= 0.100000	R07= 0.000110	R10= (used)
R02= 0.100001	R05= 0.011110	R08= 0.111110	

[continued]

R11=	1.802020218-01	A	R39=	(not used)		R67=	5.522526580-01	&
R12=	1.142424295-01	B	R40=	9.823224298-01	0	R68=	7.070000000-06	'
R13=	9.843434367-01	C	R41=	4.047114040-01	1	R69=	9.867430000-05	(
R14=	1.143434398-01	D	R42=	8.723424245-01	2	R70=	4.367980000-01	)
R15=	1.142424243-01	E	R43=	6.743424295-01	3	R71=	6.776117667-01	*
R16=	1.102020203-01	F	R44=	3.130301130-01	4	R72=	3.434983434-01	+
R17=	9.843432353-01	G	R45=	6.142424293-01	5	R73=	4.060600000-01	,
R18=	1.104040411-01	H	R46=	9.823232357-01	6	R74=	3.434343434-01	-
R19=	4.343114343-01	I	R47=	3.833302090-02	7	R75=	6.060000000-03	.
R20=	5.040439103-01	J	R48=	9.542424295-01	8	R76=	4.060980703-01	/
R21=	1.134766743-01	K	R49=	5.020202180-02	9	R77=	6.767000000-01	:
R22=	1.140404040-01	L	R50=	3.476986722-01	←	R78=	4.067670000-01	;
R23=	1.107060711-01	M	R51=	9.044449044-01	α	R79=	3.476556743-01	<
R24=	1.105345011-01	N	R52=	1.862426295-01	β	R80=	7.676767676-01	=
R25=	9.843434398-01	O	R53=	1.103030303-01	Γ	R81=	4.367557634-01	>
R26=	1.102020205-01	P	R54=	7.060116070-01	↓	R82=	7.031302050-02	?
R27=	9.843436398-01	Q	R55=	8.028432880-01	Δ	R83=	9.843464265-01	@
R28=	1.132726245-01	R	R56=	5.020209030-01	σ	R84=	1.143430000-05	[
R29=	5.424242930-02	S	R57=	3.074967430-01	◆	R85=	3.079860400-02	\
R30=	3.031103030-02	T	R58=	7.676117676-01	≠	R86=	4.343110000-01	]
R31=	9.140404091-01	U	R59=	4.184243470-02	£	R87=	6.071107060-02	↑
R32=	1.904090010-02	V	R60=	1.111111111-01	■	R88=	4.040404040-01	_
R33=	1.160706011-01	W	R61=	0.000000000		R89=	3.090300000-04	T̄
R34=	8.976347689-01	X	R62=	2.121000000-01	!	R90=	3.014041406-01	π
R35=	9.348034090-02	Y	R63=	7.070007070-02	"	R91=	1.100000000-05	|
R36=	4.383224943-01	Z	R64=	6.711671167-01	#	R92=	2.267987634-01	→
R37=	(used)		R65=	4.542114293-01	$	R93=	4.389892222-01	Σ
R38=	(used)		R66=	4.760980763-01	%	R94=	1.134343434-01	⊢

Source: Bruce Murdock (2916) (PPC CJ, V7N1P27).



CHAPTER XXIV

INTERCHANGEABLE SOLUTIONS

24-1 INTERCHANGEABLE SOLUTION ONE ("IS1"): This is a program outline, not a program. You must adapt it to your particular application; as written, it can be used to solve for any term of an equation relating 5 variables—for more terms, use LBLs F-J and Registers 06-10. Use your own output labels and prompts. NO PRINTING. To use: 1. XEQ "IS1". 2. Input values as prompted, skipping the unknown term with R/S. 3. To calculate the unknown term, press its key. 4. To change the value of a term, key in the new value, press STO, and then press its key [to change the value of D, for example, key in the new value and press STO D (= STO 04)]. Then go to step 3. Source: John Dearing (2791) (PPC CJ, V7N8P22).

A	"A"	"B"	"C"	"D"	"E"	E

```
LBL "IS1"    STO 03       (Calc. A)    (Calc. C)    (Calc. E)
SF 27        "D?"         STO 01       STO 03       STO 05
"A?"         PROMPT       "A"          "C"          "E"
PROMPT       STO 04       GTO 88       GTO 88       LBL 88
STO 01       "E?"         LBL B        LBL D        "⊢="
"B?"         PROMPT       (Calc. B)    (Calc. D)    ARCL X
PROMPT       STO 05       STO 02       STO 04       PROMPT
STO 02       CLX          "B"          "D"          END
"C?"         STOP         GTO 88       GTO 88
PROMPT       LBL A        LBL C        LBL E
```

R00: (not used)
R01: A
R02: B
R03: C
R04: D
R05: E

24-2 INTERCHANGEABLE SOLUTION TWO ("IS2"): Prints all new inputs, plus all outputs. To use: 1. XEQ "IS2". 2. Input values as prompted, skipping unknown with R/S. 3. To calculate the unknown term, press its key. 4. To change a value, reexecute the program, skipping all unchanged terms with R/S, and keying in the changed value when prompted; then go to step 3. Source: John Dearing (2791) (PPC CJ, V7N8P22).

A	"A"	"B"	"C"	"D"	"E"	E

```
LBL "IS2"    "E"          "B"          LBL E        STO IND 00
SF 27        XEQ 99       GTO 88       (Calc. E)    FS? 22
0            CLX          LBL C        STO 05       FC? 55
STO 00       STOP         (Calc. C)    "E"          RTN
"A"          LBL A        STO 03       GTO 88       ARCL X
XEQ 99       (Calc. A)    "C"          LBL 99       AVIEW
"B"          STO 01       GTO 88       CF 22        RTN
XEQ 99       "A"          LBL D        1            LBL 88
"C"          GTO 88       (Calc. D)    ST+ 00       "⊢="
XEQ 99       LBL B        STO 04       RCL IND 00   ARCL X
"D"          (Calc. B)    "D"          "⊢="         AVIEW
XEQ 99       STO 02       GTO 88       PROMPT       END
```

R00: pointer
R01: A
R02: B
R03: C
R04: D
R05: E

To modify this program to be able to store a new value by just keying in its value and then pressing its key, rather than having to reexecute the program, the following sequence for "A" is typical: [continued]

LBL A, STO 01, "A", FS?C 22, GTO 88, (Calculate A), STO 01, GTO 88.

24-3 INTERCHANGEABLE SOLUTION THREE ("IS3"): NO PRINTING. 1. XEQ "IS3". 2. Input values as prompted, skipping the unknown with R/S. 3. The unknown term will be calculated and displayed automatically. All values must be rekeyed in when the program is rerun. Source: John Dearing (2791) (PPC CJ, V7N8P22).

LBL "IS3"	XEQ 99	"B"	STO 04	"⊢ ?"	R00: storage pointer
1.1	"E"	STO 02	GTO 88	PROMPT	
STO 00	XEQ 99	GTO 88	LBL 05	STO IND 00	R01: A
CF 22	GTO IND 06	LBL 03	(Calc. E)	RCL 00	
"A"	LBL 01	(Calc. C)	"E"	FC?C 22	R02: B
XEQ 99	(Calc. A)	"C"	STO 05	STO 06	
"B"	"A"	STO 03	LBL 88	ISG 00	R03: C
XEQ 99	STO 01	GTO 88	"⊢="	END	R04: D
"C"	GTO 88	LBL 04	ARCL X		
XEQ 99	LBL 02	(Calc. D)	PROMPT		R05: E
"D"	(Calc. B)	"D"	LBL 99		R06: subroutine pointer

24-4 INTERCHANGEABLE SOLUTION FOUR ("IS4"): All inputs and outputs are printed. NO PROMPTS! 1. XEQ "IS4". 2. For each known term, key in its value, then press its key. 3. To calculate the unknown term, press its key. 4. To change any term, key in its new value, then press its key; go to step 3. 5. To recall any term, press its key (recalculates it), or press "RCL", then its key.

A	"A"	"B"	"C"	"D"	"E"	E

LBL "IS4"	(Calc. A)++	GTO 88	STO 04	FS?C 22	R00: (not used)
SF 27	STO 01	LBL C	"D"	GTO 88	
"READY"	GTO 88	STO 03	FS?C 22	(Calc. E)	R01: A
PROMPT	LBL B	"C"	GTO 88	STO 05	R02: B
LBL A	STO 02	FS?C 22	(Calc. D)	LBL 88	
STO 01	"B"	GTO 88	STO 04	"⊢="	R03: C
+	FS?C 22	(Calc. C)	GTO 88	ARCL X	R04: D
"A"	GTO 88	STO 03	LBL E	AVIEW	
FS?C 22	(Calc. B)	GTO 88	STO 05	STOP	R05: E
GTO 88	STO 02	LBL D	"E"	END	

The example shown is for an equation of 5 terms. With Labels F-J and Registers 06-10 this method will work for equations of up to 10 terms. Use your own program label and output labels. Source: John Dearing (2791) (PPC CJ, V7N8P22).

\+ Functions of the value input, such as A/100, can be calculated and stored here. If you do this, recall the value (for "A", use RCL 01) before going on.

++ Example: if the equation is A = BC/DE, use this sequence of steps to calculate A: RCL 02, RCL 03, *, RCL 04, /, RCL 05, / . Rewrite the equation for each term to be calculated.

24-5 INTERCHANGEABLE SOLUTION FIVE ("IS5"): NO PRINTING. 1. XEQ "IS5". 2. Input as prompted, skipping the unknown term with R/S. 3. The unknown term will be calculated and displayed automatically. Not practical for sensitivity analysis. Keep output labels 6 characters or fewer. NOTE: This listing is for an equation of the form A = BC/DE, which is equivalent to 1 = BC/ADE. It must be revised for an equation of another form. For terms in the numerator (B & C in this case), use XEQ a; for terms in the denominator (A, D & E here), use XEQ b. Think of a/b. Source: John Dearing (2791) (PPC CJ, V7N8P22).

[continued]

```
LBL "IS5"
CF 00
CLX
"A"
PROMPT
X=0?
XEQ b
STO 01
CLX
"B"
PROMPT
X=0?
XEQ a
STO 02
CLX
"C"
PROMPT
X=0?
XEQ a
STO 03
CLX
"D"
PROMPT
X=0?
XEQ b
STO 04
CLX
"E"
PROMPT
X=0?
XEQ b
STO 05
RCL 02
RCL 03
*
RCL 01
/
RCL 04
/
RCL 05
/
FS?C 00
1/X
CLA
ARCL 00
"├="
ARCL X
PROMPT
LBL a
SF 00
LBL b
ASTO 00
1
END
```

R00: output label
R01: A
R02: B
R03: C
R04: D
R05: E

24-6 INTERCHANGEABLE SOLUTION SIX ("IS6"): NO PRINTING; uses no numeric data registers. For use when every value can be directly converted to every other value with the proper conversion factor or routine. A can be converted to D, for example, without knowing B or C. For an example, see routine 18-21. Here, the program label is assigned to key 15 (LN). To use: 1. Key in the value to be converted. 2. XEQ "IS6" (E). 3. Press the key corresponding to the the value input. 4. Press the key corresponding to the desired output. See output. 5. For a new case, key in the value to be converted, press R/S, then go to step 3. Source: John Dearing (2791) (PPC CJ, V7N8P22).

A	A	B	C	D	"IS6"	E

```
LBL "IS6"
SF 27
"A B C D"
PROMPT
LBL A
(Convert A to D)
GTO D
LBL B
(Convert B to D)
GTO D
LBL C
(Convert C to D)
LBL D
PROMPT
LBL A
(Convert D to A)
GTO D
LBL B
(Convert D to B)
GTO D
LBL C
(Convert D to C)
LBL D
END
```

24-7 INTERCHANGEABLE SOLUTION SEVEN: To solve a system of equations with more than one unknown: for example, 5 variables, any 2 of which are unknown and 3 known, related by 5 equations; each equation relating 4 of the variables. A flag will correspond to each variable—for 5 unknowns, you might use Flags 00-04. A set flag means its corresponding variable is known; clear, it is unknown. A data register will also correspond to each variable.

Initialization: Clear all flags corresponding to variables. Also clear Flag 22, the numeric data entry flag, and set Flag 21, the printer enable flag.

Input: For each variable, have a set of steps like these: "A?", PROMPT, FS?C 22, SF 00, STO 00. The next set will have SF 01, STO 01, etc.

Body of program: Test flags to determine the knowns and/or unknowns—branch as necessary—and use the appropriate equation once you have 3 knowns (in this example). If you know which 2 variables are the unknowns, you automatically know which 3 are knowns (and vice versa). First test for 1 variable—for example, FS? 01 [is the variable corresponding to Flag 01 (B) known?]. If no, follow with 4 sets of steps, the first one of which is:

FS? 02, GTO 00, (Calculate B), STO 01, SF 01, GTO 02, LBL 00;

and the second one of which is:

FS? 03, GTO 00, (Calculate B), STO 01, SF 01, GTO 09, LBL 00, ...

If yes [Flag 01 is set (B is known)], branch to a label where two more variables are tested—say 'LBL 01, FS? 02, GTO 03, FS? 03, GTO 00....'. Following this, key in the instructions to calculate either the term corresponding to Flag 02, or the one corresponding to Flag 03. Then branch to instructions calculating the other term. LBLs 03 and 00 will lead to further flag tests, labels, and computations.

Output: Label, recall and display/print all variables. Example:

LBL 01, "A=", ARCL 00, AVIEW, "B=", ARCL 01, AVIEW, ...

Reference and source: For a good example of the use of this approach, see the "Equations of Motion" program in the HP-41C Users' Library Solutions Book, 'Physics', p. 39.

CHAPTER XXV

SYNTHETIC LOAD BYTES

25-1 SYNTHETIC LOAD BYTES PROGRAM ("LB"): "LB" is a synthetic function assembly routine. It enables the user to key up a program containing synthetic program lines simply be keying in the decimal equivalents of each byte, as determined by use of the Byte Table. Normal functions are keyed in in the ordinary manner (or can be done synthetically as well).

"LB" is included in this chapter in bar code; if the reader doesn't have a Wand, he may be able to borrow one from a friend or a dealer long enough to enter this program into memory. Saving it on magnetic cards is recommended. It can also be created using techniques found in Synthetic Programming on the HP-41C, by William C. Wickes; purchase of that book is recommended to those who wish to gain an understanding of synthetic programming. [Suggestion to dealers: having a copy of "LB" on mag cards for purchasers of this book to copy might be a friendly service.]

INSTRUCTIONS FOR USING LOAD BYTES ("LB"):

1. Load the "LB" program (XEQ SIZE 000 first if space is short), then press 'SHIFT GTO ..'. Switch to PRGM Mode and key in the first global label of your routine.

2. In PRGM Mode, key in the lines LBL "T", XEQ "LB", STOP; follow with 14 or more pluses (+). These +'s form a buffer into which the synthetic codes will be stored; the more synthetic bytes you want to load, the more +'s you'll need. RULE: To key in n synthetic bytes, the buffer should be at least 14 + 7 * INT (n/7) bytes. Extra +'s won't hurt if you have the memory; a shortcut is to key 14 +'s plus 1 '+' for each synthetic byte (in other words, for n synthetic bytes, key in 14 + n pluses).

3. Switch to RUN Mode and XEQ "T".

4. The "LB" program will prompt for a sequence of decimal byte codes (0-255). You may enter as many or as few bytes as you like, pressing R/S after each. After every seventh entry the program automatically stores the bytes.

5. To correct an immediately previous incorrect entry, just press SST to clear the prompt for byte 'n', then R/S; in a second you'll get a prompt for byte 'n-1' [this clever use of Flag 51 was suggested by Roger Hill (4940)].

6. Press R/S without a numeric entry when you don't want to load any more bytes, and the program will automatically finish the register and store it.

7. A "NO MORE" message indicates that further entry would overwrite the final END: you won't get this message if you used enough pluses. If you do get it, go to step 11.

8. Switch to PRGM Mode (you're at the third line below LBL "T") and SST several times to the first synthetic line. BST once to the previous '+', then press back-arrow as many times as necessary to delete these +'s and the STOP, XEQ "LB" and LBL "T" instructions. Now your program consists of your global label, the synthetic (or normal) lines just created with "LB", and an unknown number of +'s following. (You can SST to see these synthetic lines, then BST to get back to the global label.)

9. Begin keying in your normal program lines; when a synthetic line (or lines) is

next, just SST over it (them), then resume keying in normal lines. After the last line of the routine or program, other than the END, is keyed in (or SSTed over), press SST; if you see a '+', execute DEL 999 to clear all remaining pluses; then END your routine with 'SHIFT GTO ..' as usual. Your routine is now keyed in, complete with synthetic lines.

10. "LB" may be used wherever in a program you want the synthetic instructions to go. Perhaps you've keyed in several instructions, then you notice a synthetic line is needed. At this point, simply go to step 2 of these instructions and proceed.

11. A "NO MORE" message received in step 7 means you've run out of buffer and can't key in all your synthetic lines. Just R/S without an entry, then follow steps 8 and 9 as far as possible; when you run out of these synthetic lines, repeat the procedure as in step 10.

FUNCTION BYTE FORMATS: Row and column reference is to the Byte Table in this chapter.

1. One-byte functions: Byte 1 is from Rows 1-8 and special cases. Ex.: MEAN = 124; PROMPT = 142; the Text 0 'ultimate NOP' ("") = 240.

2. Two-byte functions: Byte 1 is from Rows 9-B and bytes 206-207; byte 2 is from the postfix part of any row (top half of the table for direct execution, bottom for indirect). Examples:

144, 111 = RCL J (111)	155, 119 = ARCL O
144, 118 = RCL N	159, 96 = TONE 6 (96)
144, 126 = RCL d	159, 109 = TONE H (109)
145, 100 = STO 00 (100)	172, 0 = FS? 00
145, 117 = STO M	173, 246 = FC? IND N
145, 245 = STO IND M	174, 112 = GTO IND T
146, 119 = ST+ O	174, 240 = XEQ IND T
150, 118 = ISG N	206, 118 = X<> N
151, 117 = DSE M	206, 127 = X<> e
152, 117 = VIEW M	206, 245 = X<> IND M
154, 117 = ASTO M	207, 117 = local LBL M

3. Alpha character strings: Byte 1 is from row F; subsequent bytes from any row (only characters from the top half of the Byte Table print; alpha strings that include any of these lower-half-of-the-table 'invisible' characters are termed nonstandard; the routine descriptions list the bytes used to create these lines).
Byte 1 from Row F determines the number of following bytes to include as part of the text string. Use 241 (Text 1) for a single character, as "A"; use 242 for a string of 2 characters, as "AB"; use 243 for 3 characters, as "ABC", and so on, up to a string of a maximum of 15 characters (255). For append character strings, byte 1 is again from Row F, and byte 2 is 127 (the append symbol). The 127 counts as a character—for example, "AB" is 242, 65, 66; "⊢AB" is 243, 127, 65, 66. Examples from routines in this book:

"⊢◆" = 242, 127, 0	"⊢×iλ◆" = 245, 127, 1, 105, 11, 0
"⊢◆◆" = 243, 127, 0, 0	"⊢◆↓****" = 247, 127, 0, 7, 42, 42, 42, 42
"⊢◆◆◆" = 244, 127, 0, 0, 0	"⊢◆◆*" = 244, 127, 0, 0, 42
"⊢◆◆◆◆" = 245, 127, 0, 0, 0, 0	"βP◆◆◆◆" = 246, 5, 80, 0, 0, 0, 0
"⊢◆◆◆◆◆" = 246, 127, 0, 0, 0, 0, 0	",x̄◆" = 243, 44, 2, 0
	"⊢◆◆A" = 244, 127, 0, 0, 65
"⊢◆◆×" = 244, 127, 0, 0, 1	"θΩ" = 242, 16, 17
"⊢θ" = 242, 127, 16	"⊢◆Δ" = 243, 127, 0, 8

4. Short-form exponents: Short-form exponents don't have the superfluous leading '1' thus saving one byte. If the number is negative, the first byte is 28 (NOT 84); the next byte is 27; if the exponent is negative, the next byte is 28; the next byte (the next 2 bytes with a 2-digit exponent) is from Row 1, Columns 0-9 of the Byte Table (bytes 16-25, equivalent to the digits 0-9, respectively) (just add 16 to the

desired digit to get its "LB" byte: if a '5' is wanted, for example, its byte number = 5+16 = 21). "E" is the equivalent of '1', but executes faster. Adjacent numbers input with "LB" must be separated with byte 0 (null) or byte 131 (ENTER).

E = 27	E-2 = 27, 28, 18	E38 = 27, 19, 24
- E = 28, 27	- E2 = 28, 27, 18	- E45 = 28, 27, 20, 21
E1 = 27, 17	- E-2 = 28, 27, 28, 18	E50 = 27, 21, 16
- E1 = 28, 27, 17	E3 = 27, 19	E99 = 27, 25, 25
E2 = 27, 18	E-4 = 27, 28, 20	E-99 = 27, 28, 25, 25

5. Global Labels: Byte 1 is 192; byte 2 is 0; byte 3 is from Row F (use a text byte 1 unit higher than the desired number of characters); byte 4 is 0; subsequent bytes from any row (only the top half of the table prints). Ex.: LBL "A#)" = 192, 0, 244, 0, 65, 35, 41; global LBL "A" = 192, 0, 242, 0, 65.

6. Global GTO or XEQ: Byte 1 is 29 or 30; byte 2 is from Row F; subsequent bytes are from any row (only the top half of the table prints). Ex.: GTO "A#)" = 29, 243, 65, 35, 41.

7. Local GTO or XEQ: For a short-form GTO, byte 1 is from Row B, byte 2 is 0; for XEQ or long-form GTO, byte 1 is from Row E or D, byte 2 is 0, byte 3 is from the postfix part of any row (direct execution only); GTO IND or XEQ IND: byte 1 is 174, byte 2 from postfix part of any row (top half of table for GTO IND, bottom for XEQ IND). Ex.: GTO 01 = 178, 0; GTO 99 = 208, 0, 99; XEQ IND 99 = 174, 227; local GTO M = 208, 0, 117.

8. Number entry: Bytes are from Row 1, Columns 0-C. Successive bytes will extend a single program line (create a multi-digit number). Use byte 0 (null) or 131 (ENTER) to terminate digit entry before starting a new program line consisting of another number. Use 28 (NOT 84) prior to digit bytes for negative numbers. Ex.: - 1.75E-10 = 28, 17, 26, 23, 21, 27, 28, 17, 16.

9. XROMS: See routines 25-3 and 25-4.

EXAMPLE: Key in routine 1-18, 'Synthetic Suspend & Reactivate Key Assignments'.

1-18 SYNTHETIC SUSPEND & REACTIVATE KEY ASSIGNMENTS ("SK" & "RK"): To suspend all system and program key assignments, key in a register pointer, 'n', then XEQ "SK"; key assignments will be stored in R'n' and R'n+1'. To reactivate these key assignments, key 'n', XEQ "RK". Minimum SIZE is n+2. Values in X, Y & Z before keying 'n' are restored. Step 24 is nonstandard; it is decimal 243, 127, 15, 255. Source: Keith Jarett (4360) (PPC ROM).

				25 X<> N
01 LBL "SK"	07 ""	13 STO N	19 ARCL IND L	26 STO ⊦
02 SIGN	08 .	14 ASTO IND L	20 "⊦◆"	27 X<> M
03 CLX	09 X<> e	15 RDN	21 ISG L	28 STO e
04 X<> ⊦	10 LBL 14	16 RTN	22 ""	29 RDN
05 XEQ 14	11 "*"	17 LBL "RK"	23 ARCL IND L	30 CLA
06 ISG L	12 X<> M	18 SIGN	*24 "⊦Φ"	31 END

---

1) If necessary, load "LB" and press 'SHIFT GTO ..'.

2) Switch to PRGM Mode.

3) Key in the first three lines of the routine: LBL "SK", SIGN, CLX.

4) Since the next step is synthetic, key in LBL "T", XEQ "LB", STOP.

5) Key in pluses to form the buffer. Examine the routine listing to count synthetic bytes:

[continued]

Line	Bytes	Byte Numbers	Line	Bytes	Byte Numbers
04 X<> ⊢	2	206, 122	22 ""	1	240
07 ""	1	240	*24 "⊢Φ"	4	243, 127, 15, 255*
09 X<> e	2	206, 127	25 X<> N	2	206, 118
12 X<> M	2	206, 117	26 STO ⊢	2	145, 122
13 STO N	2	145, 118	27 X<> M	2	206, 117
20 "⊢◆"	3	242, 127, 0	28 STO e	2	145, 127

There are 25 synthetic bytes (n=25); the buffer (number of +'s) needed is 14 + 7[INT(n/7)] = 14 + 7[INT(25/7)] = 35. (More +'s wouldn't hurt; the approximation n + 14 gives 39 pluses.) Thus, key in 35 pluses.

6) Switch out of PRGM Mode to Run Mode, then XEQ "T". See "DEC. BYTE 1.?".

7) Load the bytes, one at a time, following each with R/S. For X<> ⊢, key 206, R/S, 122, R/S; for "", key 240, R/S; for X<> e, key 206, R/S, 127, R/S. Do the same for all the remaining bytes shown in the table above (206, 117, 145, 118, etc.).

8) After the last desired instruction has been loaded (STO e = 145, R/S, 127, R/S), then R/S again without an entry. When execution stops, switch to PRGM Mode, see line 07 (+). The routine now exists in memory as:

01 LBL "SK"	05 XEQ "LB"	09 +	13 X<> M	17 "⊢Φ"	21 STO e	25 +
02 SIGN	06 STOP	10 X<> ⊢	14 STO N	18 X<> N	22 +	
03 CLX	07 +	11 ""	15 "⊢◆"	19 STO ⊢	23 +	
04 LBL "T"	08 +	12 X<> e	16 ""	20 X<> M	24 +	

and you are looking at step 07. [NOTE: This is the listing as it would print, except that X<> ᴛ is shown as it displays (X<> ⊢), and similarly, X<> [is shown as X<> M, STO \ as STO N, X<> \ as X<> N, STO ᴛ as STO ⊢, and X<> [as X<> M—see note on how Registers M, N, O, P, Q & ⊢ print and view, below.] Line 11 ("") displays as ᴛ, line 15 ("⊢◆") displays as ᴛ⊢¯, and line 17 displays as ¯▩▩.

9) Now delete steps 04-09 by SSTing to the first synthetic line (line 10), then BST once to Line 09 and press backarrow (correction) key once for each line to be deleted (6 times in this case; stop when Line 03, CLX, appears).

10) Now SST over synthetic lines and key in normal lines as you come to them. In this example, SST over the X<> ⊢, key in ISG L, SST over "", key in . (decimal—the equivalent of zero, but it executes faster), SST over X<> e, key in LBL 14 & *, SST over X<> M and STO N, key in ASTO IND L, and so on.

11) Near the end of the routine, after SSTing over line 28 (STO e) and keying in RDN and CLA, SST once more to see a '+'; execute DEL 999 to get rid of this remnant of the buffer; if you like, SST through the routine to check your work; then press 'SHIFT GTO ..'. Switch to RUN (Normal or USER) Mode. The routine is now keyed in and ready for use.

SYMBOLS FOR STATUS REGISTERS:						
Display:	M	N	O	P	Q	⊢
Printer:	[	\	]	↑	_	ᴛ

Convention used in this book: Routine listings are shown as listed by the printer, except that the status register symbols above are shown as they are displayed by the HP-41.

Source: The Synthetic Load Bytes Program ("LB") was written by Keith Jarett (4360) & William Cheeseman (4381), and appeared in the PPC Calculator Journal, V7N10P21, December 1980. Reproduced with permission.

LOAD BYTES PROGRAM LISTING:

```
01 LBL "BC"
02 XEQ 01
03 X<>Y
04 XEQ 01
05 -
06 CHS
07 RTN
08 LBL 00
09 STO M
10 "⊦***"
11 X<> M
12 X<> d
13 FS?C 04
14 SF 01
15 FS?C 05
16 SF 02
17 FS?C 06
18 SF 03
19 X<> d
20 X<> M
21 "⊦**"
22 X<> N
23 LBL 01
24 "A"
25 X<> M
26 STO N
27 ASHF
28 RDN
29 "⊦◆◆◆x"
30 RCL M
31 INT
32 LASTX
33 X<> d
34 CF 09
35 CF 10
36 CF 11
37 FS?C 14
38 SF 11
39 FS?C 15
40 SF 13
41 FS?C 16
42 SF 14
43 FS?C 17
44 SF 15
45 FS?C 18
46 SF 17
47 FS?C 19
48 SF 18
49 FS?C 20
50 SF 19
51 X<> d
52  E3
53 *
54 DEC
55 7
56 *
57 +
58  E
59 -
60 RTN
61 LBL 02
62 INT
63 OCT
64 X=0?
65 GTO 03
66 X<> d
67 4 E2
68 ST+ d
69 RDN
70 FS?C 11
71 SF 12
72 FS?C 10
73 SF 11
74 FS?C 09
75 SF 10
76 FS? 07
77 SF 09
78 FS? 06
79 SF 08
80 SF 03
81 ARCL d
82 STO d
83 "⊦**"
84 CLX
85 X<> P
86 X<> O
87 X<> N
88 STO M
89 RDN
90 RTN
91 LBL 03
92 "⊦◆"
93 RDN
94 RTN
95 LBL 04
96 XEQ 03
97 X<>Y
98 ENTER
99 X<>IND L
100 RDN
101 X<>Y
102 X<> c
103 RDN
104 RTN
105 LBL 03
106 16
107 -
108 ABS
109 RDN
110 LBL 05
111 XEQ 03
112 "⊦x"
113 X<> M
114 STO N
115 "⊦AB"
116 X<> N
117 X<> c
118 RTN
119 LBL 06
120 XEQ 03
121 X<> d
122 FS?C 07
123 SF 05
124 FS?C 08
125 SF 06
126 FS?C 09
127 SF 07
128 FS?C 10
129 SF 09
130 FS?C 11
131 SF 10
132 FS?C 12
133 SF 11
134 X<> d
135 DEC
136 RTN
137 LBL 03
138 RCL c
139 STO M
140 "⊦◆◆◆◆"
141 X<> N
142 "⊦x̄"
143 X<> M
144 X<> d
145 CF 00
146 CF 01
147 CF 02
148 CF 03
149 X<> d
150 STO N
151 RTN
152 LBL "LB"
153 CF 10
154 CF 21
155 RCL b
156 XEQ 00
157  E
158 -
159 7
160 /
161 INT
162 XEQ 06
163  E3
164 /
165 +
166 FIX 0
167  E
168 LBL 08
169 7 E-3
170 +
171 SF 22
172 DSE Y
173 GTO 10
174 "NO MORE"
175 AVIEW
176 TONE 6
177 RTN
178 LBL 09
179 RDN
180 LBL 10
181 FC?C 22
182 GTO 03
183 RCL M
184 "DEC. BYTE "
185 ARCL Y
186 "⊦?"
187 AVIEW
188 CLA
189 STO M
190 RDN
191 TONE 7
192 STOP
193 FS? 51
194 GTO 11
195 LBL 03
196 FC? 22
197 .
198 XEQ 02
199 ISG X
200 GTO 10
201 RCL M
202 RCL Z
203 XEQ 04
204 STO M
205 X<> L
206 16
207 +
208 X<>Y
209 FS? 22
210 GTO 08
211 RTN
212 LBL 11
213 RCL M
214 CLA
215 STO M
216 ASTO M
217 RDN
218  E
219 -
220 ENTER
221 SF 22
222 7
223 MOD
224 INT
225 X>0?
226 GTO 09
227 RDN
228 7 E-3
229 -
230 ISG Y
231 ""
232 GTO 10
233 END
```

(441 bytes)

SYNTHETIC LOAD BYTES PROGRAM By Keith Jarett (4360) & William Cheeseman (4381), PPC Calculator Journal, V7N10P21, December 1980.

ROW 1 (1 : 4)

ROW 2 (5 : 11)

ROW 3 (12 : 18)

ROW 4 (18 : 24)

ROW 5 (24 : 29)

ROW 6 (30 : 37)

ROW 7 (37 : 43)

ROW 8 (44 : 50)

ROW 9 (50 : 60)

ROW 10 (61 : 68)

ROW 11 (69 : 75)

ROW 12 (76 : 82)

ROW 13 (82 : 88)

ROW 14 (88 : 96)

ROW 15 (97 : 106)

ROW 16 (107 : 114)

ROW 17 (114 : 120)

ROW 18 (120 : 126)

ROW 19 (127 : 133)

ROW 20 (133 : 140)

ROW 21 (140 : 145)

ROW 22 (145 : 152)

ROW 23 (152 : 156)

ROW 24 (156 : 165)

ROW 25 (166 : 172)

ROW 26 (173 : 176)

ROW 27 (177 : 184)

ROW 28 (184 : 186)

ROW 29 (187 : 195)

ROW 30 (196 : 202)

ROW 31 (202 : 209)

ROW 32 (209 : 217)

ROW 33 (218 : 228)

ROW 34 (228 : 233)

HP-41 BYTE TABLE (HEX TABLE)

1-BYTE INSTRUCTIONS (Rows 0–7) — 2-BYTE (Rows 8–B) — 3-BYTE (Rows C–E) — ? (Row F)

POSTFIX DIRECT (Rows 0–7) — POSTFIX INDIRECT (Rows 8–F)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0 ◆ NULL 00	1 × LBL 00 01	2 x̄ LBL 01 02	3 ← LBL 02 03	4 α LBL 03 04	5 β LBL 04 05	6 Γ LBL 05 06	7 ↓ LBL 06 07	8 Δ LBL 07 08	9 σ LBL 08 09	10 ◆ LBL 09 10	11 λ LBL 10 11	12 μ LBL 11 12	13 ∡ LBL 12 13	14 τ LBL 13 14	15 Φ LBL 14 15	0
1	16 θ 0 16	17 Ω 1 17	18 δ 2 18	19 Å 3 19	20 å 4 20	21 Ä 5 21	22 ä 6 22	23 Ö 7 23	24 ö 8 24	25 Ü 9 25	26 ü . 26	27 Æ EEX 27	28 æ CHS 28	29 ≠ GTO α 29	30 £ XEQ α 30	31 ▒ SPARE 31	1
2	32 RCL 00 32 SPACE	33 ! RCL 01 33	34 " RCL 02 34	35 # RCL 03 35	36 $ RCL 04 36	37 % RCL 05 37	38 & RCL 06 38	39 ' RCL 07 39	40 (RCL 08 40	41) RCL 09 41	42 * RCL 10 42	43 + RCL 11 43	44 , RCL 12 44	45 - RCL 13 45	46 . RCL 14 46	47 / RCL 15 47	2
3	48 0 STO 00 48	49 1 STO 01 49	50 2 STO 02 50	51 3 STO 03 51	52 4 STO 04 52	53 5 STO 05 53	54 6 STO 06 54	55 7 STO 07 55	56 8 STO 08 56	57 9 STO 09 57	58 : STO 10 58	59 ; STO 11 59	60 < STO 12 60	61 = STO 13 61	62 > STO 14 62	63 ? STO 15 63	3
4	64 @ + 64	65 A - 65	66 B * 66	67 C / 67	68 D X<Y? 68	69 E X>Y? 69	70 F X<=Y? 70	71 G Σ+ 71	72 H Σ- 72	73 I HMS+ 73	74 J HMS- 74	75 K MOD 75	76 L % 76	77 M %CH 77	78 N P-R 78	79 O R-P 79	4
5	80 P LN 80	81 Q X↑2 81	82 R SQRT 82	83 S Y↑X 83	84 T CHS 84	85 U E↑X 85	86 V LOG 86	87 W 10↑X 87	88 X E↑X-1 88	89 Y SIN 89	90 Z COS 90	91 [TAN 91	92 \ ASIN 92	93] ACOS 93	94 ↑ ATAN 94	95 _ DEC 95	5
6	96 ‛ 1/X 96	97 a ABS 97	98 b FACT 98	99 c X≠0? 99	100 d X>0? 00 100	101 e LN1+X 01 101	102 f X<0? A 102	103 g X=0? B 103	104 h INT C 104	105 i FRC D 105	106 j D-R E 106	107 k R-D F 107	108 l HMS G 108	109 m HR H 109	110 n RND I 110	111 o OCT J 111	6
7	112 p CLΣ T	113 q X<>Y Z	114 r PI Y	115 s CLST X	116 t R↑ L	117 u RDN M [	118 v LASTX N \	119 w CLX O]	120 x X=Y? P ↑	121 y X≠Y? Q _	122 z SIGN ├ T	123 π X<=0? a	124 | MEAN b	125 → SDEV c	126 Σ AVIEW d	127 ├ CLD e	7
8	128 ◆ DEG 00	129 × RAD 01	130 x̄ GRAD 02	131 ← ENTER 03	132 α STOP 04	133 β RTN 05	134 Γ BEEP 06	135 ↓ CLA 07	136 Δ ASHF 08	137 σ PSE 09	138 ◆ CLRG 10	139 λ AOFF 11	140 μ AON 12	141 ∡ OFF 13	142 τ PROMPT 14	143 Φ ADV 15	8
9	144 θ RCL 16	145 Ω STO 17	146 δ ST+ 18	147 Å ST- 19	148 å ST* 20	149 Ä ST/ 21	150 ä ISG 22	151 Ö DSE 23	152 ö VIEW 24	153 Ü ΣREG 25	154 ü ASTO 26	155 Æ ARCL 27	156 æ FIX 28	157 ≠ SCI 29	158 £ ENG 30	159 ▒ TONE 31	9
A	160 XROM 32	161 ! XROM 33	162 " XROM 34	163 # XROM 35	164 $ XROM 36	165 % XROM 37	166 & XROM 38	167 ' XROM 39	168 (SF 40	169) CF 41	170 * FS?C 42	171 + FC?C 43	172 , FS? 44	173 - FC? 45	174 . GTO/XEQ 46 IND*	175 / SPARE 47	A
B	176 0 SPARE 48	177 1 GTO 00 49	178 2 GTO 01 50	179 3 GTO 02 51	180 4 GTO 03 52	181 5 GTO 04 53	182 6 GTO 05 54	183 7 GTO 06 55	184 8 GTO 07 56	185 9 GTO 08 57	186 : GTO 09 58	187 ; GTO 10 59	188 < GTO 11 60	189 = GTO 12 61	190 > GTO 13 62	191 ? GTO 14 63	B
C	192 @ GLOBAL 64	193 A GLOBAL 65	194 B GLOBAL 66	195 C GLOBAL 67	196 D GLOBAL 68	197 E GLOBAL 69	198 F GLOBAL 70	199 G GLOBAL 71	200 H GLOBAL 72	201 I GLOBAL 73	202 J GLOBAL 74	203 K GLOBAL 75	204 L GLOBAL 76	205 M GLOBAL 77	206 N X<> 78	207 O LBL 79	C
D	208 P GTO 80	209 Q GTO 81	210 R GTO 82	211 S GTO 83	212 T GTO 84	213 U GTO 85	214 V GTO 86	215 W GTO 87	216 X GTO 88	217 Y GTO 89	218 Z GTO 90	219 [GTO 91	220 \ GTO 92	221] GTO 93	222 ↑ GTO 94	223 _ GTO 95	D
E	224 ‛ XEQ 96	225 a XEQ 97	226 b XEQ 98	227 c XEQ 99	228 d XEQ 00 100	229 e XEQ 01 101	230 f XEQ A 102	231 g XEQ B 103	232 h XEQ C 104	233 i XEQ D 105	234 j XEQ E 106	235 k XEQ F 107	236 l XEQ G 108	237 m XEQ H 109	238 n XEQ I 110	239 o XEQ J 111	E
F	240 p TEXT 0 T	241 q TEXT 1 Z	242 r TEXT 2 Y	243 s TEXT 3 X	244 t TEXT 4 L	245 u TEXT 5 M [	246 v TEXT 6 N \	247 w TEXT 7 O]	248 x TEXT 8 P ↑	249 y TEXT 9 Q _	250 z TEXT 10 ├ T	251 π TEXT 11 a	252 | TEXT 12 b	253 → TEXT 13 c	254 Σ TEXT 14 d	255 ├ TEXT 15 e	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

* Byte 174 is GTO IND if the byte following is from the top half of the table (Rows 0-7); it is XEQ IND if the byte following is from the bottom half of the table (Rows 8-F).

SAMPLE BYTE TABLE BOX: For a detailed description of the Byte Table, see Section 2B, 'The Byte Table' (pp 9-16), in Synthetic Programming on the HP-41C, by William C. Wickes.

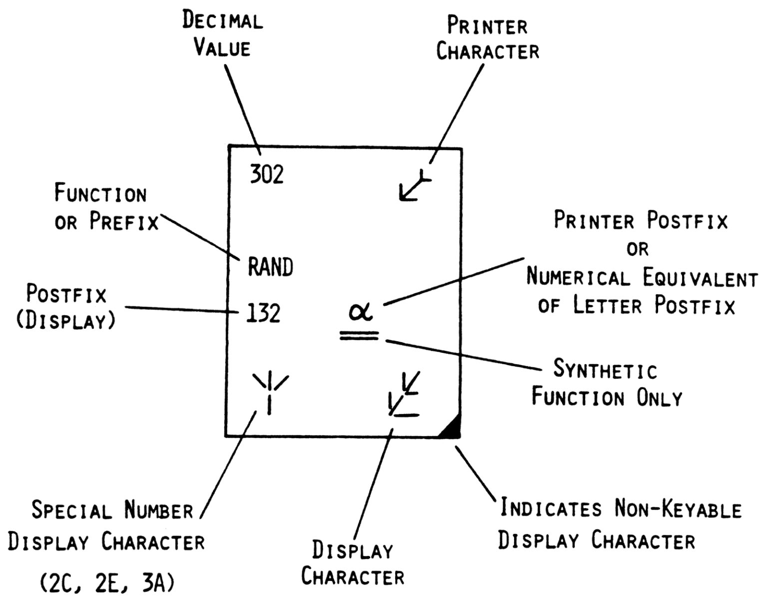


FIGURE 2-4. SAMPLE BYTE TABLE "BOX"
Source: Syn. Prog. on the HP-41C, By William C. Wickes

25-2 KEYING PROGRAMS WITH SEVERAL SYNTHETICS: A good procedure to use when keying up a program containing several synthetic lines is to load them all at once in the order that they occur in the program, and simply SST over them as needed when keying the program into memory ahead of the 'synthetic group' of instructions. Source: Richard Nelson (1) (PPC Calculator Journal Members Newsletter, V7N10).

25-3 COMPUTE XROM "KA" & "LB" INPUTS ("XR"): This routine will figure out the decimal inputs to use for assigning, or creating in program memory, functions contained in the ROM of any HP-41 peripheral or module. The input to the routine is the function's XROM number in the form AA.BB (just as it appears as XROM AA,BB). The output is the two decimal inputs needed in "KA" [any key assignments program, such as the one in Synthetic Programming on the HP-41C (pp 44-47, 86-87)] or "LB", in the form AAA.BBB. The assignment or program step produced is for 'real' XROM, not pseudo XROM such as those produced with synthetic commands like tones. When the proper module or peripheral is connected, these assignments will work as if they had been normally assigned or keyed into program memory with the module or peripheral connected. Without the module or peripheral, pressing its assigned key or executing it in a program gives "NONEXISTENT". Example: for the printer function BLDSPEC (XROM 29,06), key in 29.06, XEQ "XR"; see 167.070; hence the two bytes to use in "KA" or "LB" are 167 and 70. Source: David Bartholomew (3666) (PPC CJ, V7N7P10).

01 LBL "XR"	06 /	11 160	16 256	21 +	26 END
02 FRC	07 INT	12 +	17 *	22 1 E3	
03 LASTX	08 LASTX	13 X<>Y	18 X<>Y	23 /	
04 INT	09 STO T	14 RCL Z	19 1 E2	24 +	
05 4	10 RDN	15 FRC	20 *	25 FIX 3	(44 bytes)

25-4 SYNTHETIC "XROM" INPUTS FOR "LB" (LOAD BYTES PROGRAM) ("XL"): This routine allows the user to create program instructions for any XROM instruction. If the two XROM numbers are represented by 'A' and 'B', then A, ENTER, B, XEQ "XL" produces the first byte in X and the second in Y. For example, to find the bytes to put "ACSPEC" (XROM 29,04) into a program using "LB", key 29, ENTER 4, XEQ "XL"; output is '167' in X and '68' in Y; then use bytes 167, 68 with "LB" to put ACSPEC into a program. Works whether a printer is plugged in or not. Source: Roger Hill (4940) (PPC ROM).

```
01 LBL "XL"
02 X<>Y
03 640
04 +
05 64
06 *
07 +
08 256
09 XEQ "QR"
10 X<>Y
11 END
            (26 bytes)
```

```
01 LBL "QR"
02 X<>Y
03 STO O
04 X<>Y
05 MOD
06 ST- O
07 LASTX
08 ST/ O
09 CLX
10 X<> O
11 X<>Y
12 END      (23 bytes)
```

25-5 SYNTHETIC "BLDSPEC" INPUTS FOR "LB" ("BL"): This routine processes the seven "BLDSPEC" numbers (column print numbers) to produce the seven "LB" bytes. Remember to preceed these seven text bytes with a Text 7 byte (247). 1. Key the first BLDSPEC number, XEQ "BL"; see the first "LB" byte. 2. Key the second BLDSPEC number, press R/S; see the second "LB" byte. 3. Repeat step 2 for the remaining BLDSPEC numbers. 247 (the Text 7 byte) followed by the seven bytes just generated are the eight bytes to use with "LB" to create the appropriate text line. In a program, follow this text line with RCL M, ACSPEC (bytes 144, 117; 167, 68) to put the character into the printer buffer. EXAMPLE: Use "BL" to compute the bytes needed for the arrow symbol in routine 22-18, Solution: the BLDSPEC numbers are 120, 96, 80, 72, 7, 6, 4. Key 120, XEQ "BL", see '17'; key 96, R/S, see '227'; likewise, key the remaining numbers, following each with R/S to find the corresponding bytes.

BLDSPEC:	120	96	80	72	7	6	4
"LB" BYTES:	17	227	5	9	1	195	4

Thus, to key the synthetic BLDSPEC text line into a program, use the following bytes with "LB": 247, 17, 227, 5, 9, 1, 195, 4. Follow the text line with RCL M, ACSPEC. Source: Roger Hill (4940) (PPC ROM). See 22-18.

```
01 LBL "BL"
02 2
03 STO M
04 X↑2
05 X↑2
06 X<>Y
07 LBL 02
08 128
09 RCL M
10 ST+ M
11 /
12 XEQ "QR"
13 RCL M
14 *
15 X<> Z
16 +
17 STOP
18 X<>Y
19 RDN
20 GTO 02
21 END
            (39 bytes)
```

```
01 LBL "QR"
02 X<>Y
03 STO O
04 X<>Y
05 MOD
06 ST- O
07 LASTX
08 ST/ O
09 CLX
10 X<> O
11 X<>Y
12 END
            (23 bytes)
```

25-6 SYNTHETIC FLAG INPUTS FOR "LB" ("FL"): Given the flags to be set, this routine will output the bytes to be loaded with "LB" to create the synthetic text line that will set or clear all 56 flags in one operation. See routine 6-8, Synthetic Mass Flag Control. The use of a synthetic text line to control flags is memory efficient only if seven or more flags are to be set or cleared. To determine flag input bytes for "LB", XEQ "FL" (see zero); key the first flag to be set. If a tone sounds, the display shows a byte; press R/S for another output. When the number displayed is negative (the number of the last flag input), key the next flag and R/S. Repeat. When all flags to be set are input, the last input, if Flag 55 was not set, should be '56'. The seven outputs provide the seven decimal inputs to load with "LB". Remember to precede the seven bytes with '247' (the Text 7 byte).

Example: the following flags are to be set during initialization of a program; determine the bytes to input to "LB" to set them with a synthetic text line: Flags 5, 25, 26, 28, 39, 40, 44. Solution:

DO	SEE	RESULT
XEQ "FL"	0.00	(start of program)
5, R/S	-5.00	Negative — key next flag
25, R/S	4.00	Tone: first byte
R/S	0.00	Tone: second byte
R/S	0.00	Tone: third byte
R/S	-25.00	Negative — key next flag
26, R/S	-26.00	Negative — key next flag
28, R/S	-28.00	Negative — key next flag
39, R/S	104.00	Tone: fourth byte
R/S	1.00	Tone: fifth byte
R/S	-39.00	Negative — key next flag
40, R/S	-40.00	Negative — key next flag
44, R/S	-44.00	Negative — all desired flags input
56, R/S	136.00	Tone: sixth byte
R/S	0.00	Tone: seventh byte
R/S	-56.00	(end of program)

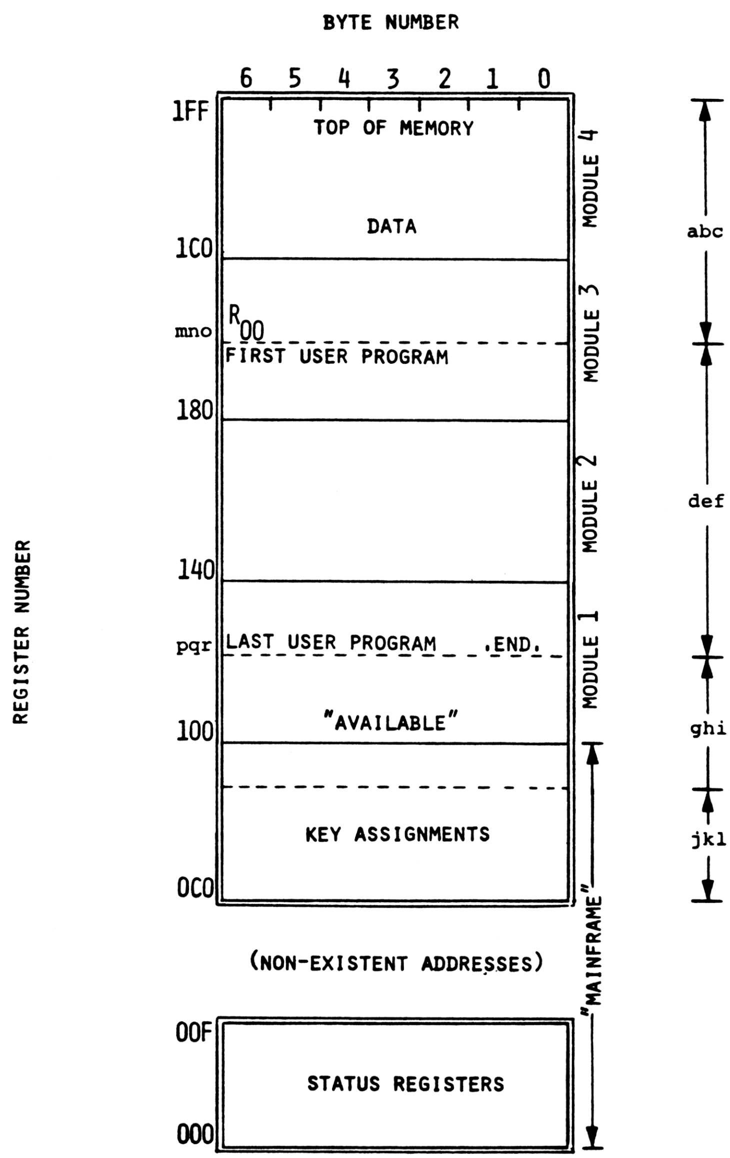
Thus, the bytes to be input to "LB", including the Text 7 byte (247), are: 247, 4, 0, 0, 104, 1, 136, 0. Follow this synthetic flag text line with RCL M, STO d (which can also be created with "LB" using the bytes 144, 117; 145, 126). NOTE: All flags not specifically set are cleared. Source: Keith Jarett (4360) (PPC ROM). See 6-8.

```
01 LBL "FL"     13 X<> M      25 CHS        37 ""              01 LBL "QR"
02 CLA          14 7          26 GTO 00     38 TONE 7          02 X<>Y
03 CLST         15 X<>Y       27 LBL 13     39 STOP            03 STO O
04 LBL 00       16 -          28 X<> M      40 END             04 X<>Y
05 STOP         17 2          29 ENTER                         05 MOD
06 RCL X        18 X<>Y       30 XEQ 14                        06 ST- O
07 8            19 Y↑X        31 RDN                           07 LASTX
08 XEQ "QR"     20 ST+ N      32 GTO 01                        08 ST/ O
09 LBL 01       21  E         33 LBL 14                        09 CLX
10 X<> M        22 X=Y?       34 CLX                           10 X<> O
11 X≠Y?         23 XEQ 14     35 X<> N                         11 X<>Y
12 GTO 13       24 R↑         36 ISG M      (65 bytes)         12 END
```

CHAPTER XXVI

REFERENCE

26-1 HP-41 USER MEMORY PARTIONING: Source: 'Synthetic Programming on the HP-41C', by William C. Wickes.



26-2 THE STATUS REGISTERS: Source: 'Synthetic Programming on the HP-41C', by William C. Wickes.

BYTE NUMBER

NAME	6	5	4	3	2	1	0	ADDRESS
e	SHIFTED KEY ASSIGNMENTS				SCRATCH		LINE NUMBER	00F
d	USER FLAGS				SYSTEM FLAGS			00E
c	ΣREG		SCRATCH	COLD START	R_{00}		.END.	00D
b	3RD RETURN	2ND RETURN		1ST RETURN		ADDRESS POINTER		00C
a	6TH RETURN		5TH RETURN		4TH RETURN		3RD RETURN	00B
⊢	UNSHIFTED KEY ASSIGNMENTS				SCRATCH			00A
Q	TEMPORARY ALPHA SCRATCH							009
P	SCRATCH (ALPHA REGISTER 25-28)				ALPHA REGISTER 22-24			008
O	ALPHA REGISTER 15-21							007
N	ALPHA REGISTER 8-14							006
M	ALPHA REGISTER 1-7							005
L	STACK L							004
X	STACK X							003
Y	STACK Y							002
Z	STACK Z							001
T	STACK T							000
	SIGN	MANTISSA					SIGN / EXPONENT	

26-3 STACK REARRANGEMENTS: This listing gives all 256 rearrangements of the stack; all but 22 are 3 instructions or fewer. Source: John Dearing (2791) (PPC CJ, V7N2P22). See 7-11.

...... T ......

TTTT	R↑, ENTER, ENTER, ENTER
or	R↑, ENTER, ENTER, STO T
TTTX	R↑, ENTER, ENTER
or	R↑, RCL X, RCL X
TTTY	R↑, ENTER, STO Z
TTTZ	R↑, STO Y, STO Z
TTXT	R↑, ENTER, STO T
TTXX	STO Y, R↑, ENTER
or	STO Y, R↑, RCL X
TTXY	R↑, RCL X
TTXZ	X<>Y, R↑, STO Y
TTYT	R↑, STO Y, STO T
TTYX	X<>Y, R↑, ENTER
or	X<>Y, R↑, RCL X
TTYY	RDN, RCL X, R↑, ENTER
or	RDN, RCL X, R↑, RCL X
TTYZ	R↑, STO Y
TTZT	RDN, RCL Z, STO Y
TTZX	X<> T, STO Y
TTZY	RDN, X<> Z, ENTER
or	RDN, X<> Z, RCL X
TTZZ	X<> T, STO Y, RCL Z, RDN

TYTT	RDN, RCL Z, STO Z
TYTX	X<> T, STO Z
TYTY	X<> T, RCL Y, RCL Y
TYTZ	X<>Y, R↑, STO Z
TYXT	X<>Y, R↑, STO T
TYXX	STO Z, X<> T
TYXY	X<>Y, STO Z, R↑
TYXZ	X<>Y, R↑
TYYT	RDN, STO Y, RCL Z
TYYX	RDN, STO Y, X<> Z
TYYY	R↑, RCL Z, STO Z, RDN
TYYZ	X<>Y, STO Y, R↑
TYZT	RDN, RCL Z
TYZX	X<> T
TYZY	RDN, ENTER, X<> T
TYZZ	RDN, RCL Y, X<> T

TXTT	R↑, STO Z, STO T
TXTX	R↑, RCL Y, RCL Y
TXTY	X<>Y, X<> T, STO Z
TXTZ	R↑, STO Z
TXXT	STO Y, R↑, STO T
TXXX	STO Z, STO Y, R↑
TXXY	R↑, RCL Y, X<>Y
TXXZ	STO Y, R↑
TXYT	R↑, STO T
TXYX	STO Z, R↑
TXYY	R↑, RCL Z, RDN
TXYZ	R↑
TXZT	X<>Y, RDN, RCL Z
TXZX	STO Y, X<> T
TXZY	X<>Y, X<> T
TXZZ	RDN, RDN, STO T, RDN

TZTT	RDN, RCL Z, STO Y, RDN
TZTX	RDN, RDN, RCL Y
TZTY	X<> T, RCL Z, RCL Y
TZTZ	RDN, X<> Z, RCL Y, RCL Y
TZXT	X<> T, STO Y, RDN
TZXX	STO Y, RDN, X<> Z
TZXY	RDN, RDN, X<>Y
TZXZ	X<> Z, STO Y, X<> T
TZYT	RDN, X<>Y, RCL Z
TZYX	RDN, X<> Z
TZYY	RDN, STO T, X<> Z
TZYZ	RDN, RCL Y, R↑
TZZT	RDN, RDN, RCL X, RCL Z
TZZX	X<> Z, STO Y, R↑
TZZY	RDN, X<>Y, RCL X, R↑
TZZZ	X<> T, RCL Z, STO Z, RDN

...... X ......

XTTT	R↑, STO Z, STO T, RDN
XTTX	R↑, RCL X, RCL Z
XTTY	R↑, ENTER, X<> Z
XTTZ	R↑, STO Z, X<>Y
XTXT	R↑, STO Z, RCL Y
XTXX	R↑, RCL Y, STO T
XTXY	R↑, RCL Y
XTXZ	R↑, X<>Y, STO Z
XTYT	R↑, STO T, X<>Y
XTYX	X<>Y, R↑, RCL Z
XTYY	R↑, RCL Z, X<> Z
XTYZ	R↑, X<>Y
XTZT	R↑, STO Z, RDN
XTZX	STO Y, X<> T, X<>Y
XTZY	RDN, RDN, X<> Z
XTZZ	X<> Z, STO Y, RDN, RDN

XYTT	R↑, STO T, RDN
XYTX	STO Z, RDN, X<>Y
XYTY	X<>Y, STO Z, RDN
XYTZ	X<> Z, RDN, X<>Y
XYXT	STO Z
XYXX	STO T, STO Z
XYXY	RCL Y, RCL Y
XYXZ	X<>Y, RCL Y
XYYT	X<>Y, STO Z, X<>Y
XYYX	X<>Y, RCL X, RCL Z
XYYY	RCL Y, STO T, X<>Y
XYYZ	RCL Y, X<>Y
XYZT	(original order)
XYZX	STO T
XYZY	RCL Y, RDN
XYZZ	RCL Z, RDN

XXTT	R↑, STO Z, X<>Y, ENTER
or	R↑, STO Z, X<>Y, RCL X
XXTX	STO Y, RDN, STO Y
XXTY	X<>Y, RDN, STO Y
XXTZ	STO Y, R↑, X<> Z
XXXT	STO Y, STO Z
XXXX	ENTER, ENTER, ENTER
or	ENTER, ENTER, STO T
XXXY	RCL X, RCL X
XXXZ	ENTER, STO Z
XXYT	STO Z, X<>Y, X<> Z
XXYX	ENTER, STO T
XXYY	RCL Y, X<>Y, ENTER
or	RCL Y, X<>Y, RCL X
XXYZ	RCL X
XXZT	STO Y
XXZX	STO Y, STO T
XXZY	RCL Z, X<>Y, ENTER
or	RCL Z, X<>Y, RCL X
XXZZ	RCL Z, RDN, STO Y

XZTT	X<>Y, RDN, RCL Z, RDN
XZTX	STO Y, RDN
XZTY	X<>Y, RDN
XZTZ	X<>Y, RDN, RCL Y, RDN
XZXT	STO Y, X<> Z, X<>Y
XZXX	STO Y, RDN, STO Z
XZXY	RCL Z, RCL Y
XZXZ	RCL Z, X<>Y, STO Z
XZYT	RDN, X<>Y, R↑
XZYX	X<> Z, RCL Z
XZYY	X<>Y, STO T, RDN
XZYZ	RCL Z, X<>Y
XZZT	X<> Z, STO Y, X<> Z
XZZX	RCL Z, RCL X, RCL Z
XZZY	RCL Z, ENTER, X<> Z
XZZZ	RCL Z, STO Z, X<>Y

...... Y ......

YTTT	RDN, RCL Z, STO Z, RDN
YTTX	R↑, RCL X, R↑
YTTY	R↑, STO Y, RCL Z
YTTZ	X<> T, ENTER, X<> Z
YTXT	R↑, ENTER, X<> T
YTXX	R↑, RCL Y, X<> T
YTXY	R↑, RCL Z
YTXZ	RDN, X<>Y, RDN
YTYT	X<> T, STO Z, RCL Y
YTYX	X<>Y, R↑, RCL Y
YTYY	R↑, RCL Z, STO Z
YTYZ	X<> T, RCL Y
YTZT	RDN, RCL Z, X<>Y
YTZX	X<> T, X<>Y
YTZY	RDN, X<> Z, RCL Z
YTZZ	X<> T, RCL Z, X<> Z

YXTT	R↑, STO T, X<> Z
YXTX	STO Z, RDN
YXTY	STO Z, RDN, STO T
YXTZ	X<> Z, RDN
YXXT	STO Z, X<>Y
YXXX	RCL X, RCL X, R↑
YXXY	RCL X, RCL Z
YXXZ	ENTER, X<> Z
YXYT	X<>Y, STO Z
YXYX	STO Z, RCL Y
YXYY	RCL Y, STO T
YXYZ	RCL Y
YXZT	X<>Y
YXZX	STO T, X<>Y
YXZY	X<>Y, STO T
YXZZ	RCL Z, X<> Z

YYTT	RDN, STO Y, RCL Z, RDN
YYTX	RDN, STO Y
YYTY	RDN, STO Y, STO T
YYTZ	X<> Z, RDN, STO Y
YYXT	STO Z, RDN, ENTER
or	STO Z, RDN, RCL X
YYXX	STO Z, X<>Y, ENTER
or	STO Z, X<>Y, RCL X
YYXY	RCL Y, RCL Z
YYXZ	X<>Y, RCL X
YYYT	RDN, ENTER, STO Z
YYYX	X<>Y, ENTER, ENTER
or	X<>Y, RCL X, RCL X
YYYY	RCL Y, ENTER, STO Z
YYYZ	RCL Y, STO Y
YYZT	RDN, RCL X
YYZX	RCL Y, X<>Y, RDN
YYZY	RDN, ENTER, STO T
YYZZ	RCL Z, X<> Z, STO Y

YZTT	RDN, RCL Z, RDN
YZTX	RDN
YZTY	RDN, STO T
YZTZ	RDN, RCL Y, RDN
YZXT	X<> Z, X<>Y
YZXX	STO T, RDN
YZXY	RCL Z, RCL Z
YZXZ	X<>Y, RCL Z, X<>Y
YZYT	RDN, X<>Y, RCL Y
YZYX	RDN, STO Z
YZYY	RDN, STO Z, STO T
YZYZ	RDN, RCL Y, RCL Y
YZZT	RDN, RCL Y, X<>Y
YZZX	RCL Z, RCL X, R↑
YZZY	RCL Z, STO Y, RCL Z
YZZZ	RCL Z, STO Y, X<> Z

...... Z ......

ZTTT	R↑, STO Y, STO Z, R↑
ZTTX	X<> T, STO Y, X<> Z
ZTTY	R↑, STO Y, R↑
ZTTZ	X<> T, STO Y, RCL Z
ZTXT	R↑, STO Z, R↑
ZTXX	STO Y, RDN, RDN
ZTXY	RDN, RDN
ZTXZ	RDN, RDN, STO T
ZTYT	R↑, STO Y, X<> T
ZTYX	X<>Y, RDN, RDN
ZTYY	RDN, STO T, RDN
ZTYZ	RDN, RCL Z, RCL Z
ZTZT	RDN, RDN, RCL Y, RCL Y
ZTZX	X<> Z, STO Y, X<> T, R↑
ZTZY	RDN, RDN, STO Z
ZTZZ	X<> T, RCL Z, STO Z

ZXTT	R↑, STO Z, X<> T
ZXTX	STO Y, RDN, X<>Y
ZXTY	X<>Y, RDN, X<>Y
ZXTZ	X<> Z, STO Y, RDN
ZXXT	STO Y, X<> Z
ZXXX	ENTER, STO Z, R↑
ZXXY	RCL X, R↑
ZXXZ	STO Y, RCL Z
ZXYT	X<>Y, X<> Z
ZXYX	ENTER, X<> T
ZXYY	RCL Y, X<> T
ZXYZ	RCL Z
ZXZT	X<>Y, RDN, RCL Y
ZXZX	ENTER, X<> T, STO Z
ZXZY	RCL Z, X<>Y, RCL Y
ZXZZ	RCL Z, STO Z

ZYTT	RDN, RCL Z, X<> Z
ZYTX	RDN, X<>Y
ZYTY	RDN, STO T, X<>Y
ZYTZ	RDN, X<>Y, STO T
ZYXT	X<> Z
ZYXX	STO T, X<> Z
ZYXY	RCL Y, R↑
ZYXZ	X<>Y, RCL Z
ZYYT	RDN, ENTER, X<> Z
ZYYX	RDN, STO Z, X<>Y
ZYYY	RCL Y, STO Y, R↑
ZYYZ	X<>Y, STO Y, RCL Z
ZYZT	RDN, RCL Y
ZYZX	RDN, X<>Y, STO Z
ZYZY	RDN, STO Z, RCL Y
ZYZZ	X<>Y, RCL Z, STO Z

ZZTT	R↑, STO Y, RCL T, ENTER
or	R↑, STO Y, RCL T, RCL X
ZZTX	RDN, RDN, ENTER
or	RDN, RDN, RCL X
ZZTY	X<> T, RCL Z, ENTER
or	X<> T, RCL Z, RCL X
ZZTZ	RDN, RDN, STO Z, ENTER
or	RDN, RDN, STO Z, RCL X
ZZXT	X<> Z, STO Y
ZZXX	STO Y, RCL Z, RCL X
ZZXY	RCL Z, R↑
ZZXZ	X<>Y, RCL Z, STO Y
ZZYT	RDN, X<>Y, ENTER
or	RDN, X<>Y, RCL X
ZZYX	X<> Z, RCL X
ZZYY	RCL Y, X<> T, STO Y
ZZYZ	RCL Z, STO Y
ZZZT	RDN, RCL Y, STO Y
ZZZX	RCL Z, ENTER, ENTER
or	RCL Z, RCL X, RCL X
ZZZY	RCL Z, ENTER, STO Z
ZZZZ	RCL Z, STO Y, STO Z

26-4 HP-41 FLAG TABLE: Source: PPC Journal (V6N5P27) & HP KEY NOTES (V4N3P5).

FLAG NO.	FLAG NAME	IF SET (OR SET BY)	STATUS AT TURN-ON*
		F U L L - U S E F L A G S	
00-10	General Purpose	00-04 annunciators.	M,1
11	Automatic Execution	Program execution starts at turn-on.	C
12	Printer Double Wide	Prints all double wide.	C
13	Printer Lowercase	Alphabetics in lowercase letters.	C
14	Card Reader Overwrite	Writes on cards with clipped corners.	C
15	Future use		
16	Future use		
17	Future use		
18	Future use		
19	Future use		
20	Future use		
21	Printer Enable	Flag 55 usually set; print if possible.	2
22	Numeric Input	Numeric data entry sets flag.	C
23	Alpha Input	Alpha data entry sets flag.	C
24	Range Error Ignore	Range Error ignored.	C
25	Error Ignore	Operation not performed, flag cleared.	C
26	Audio Enable	Tones audible.	S
27	User Mode	USER Mode.	M,1
28	Decimal vs Comma	Radix is decimal point.	M,3
29	Digit Grouping	Digit grouping (in groups of three) if set; does not show radix in Fix 0 if clear.	M,3

T E S T - O N L Y F L A G S

Set = YES, Clear = NO; shows "NONEXISTENT" if set or clear is attempted.

FLAG NO.	FLAG NAME	IF SET (OR SET BY)	STATUS AT TURN-ON*
30	Catalog	Executing a catalog. Always tests clear.	C
31-35	Peripheral	Peripheral connected.	M
		# Digits: 0 1 2 3 4 5 6 7 8 9	
36	No. Digits Displayed	C C C C C C C C S S	M,1
37	" " "	C C C C S S S S C C	M,3
38	" " "	C C S S C C S S C C	M,1
39	" " "	C S C S C S C S C S	M,1
40	Display Format	FIX display } SCI display if both clear.	M,1
41	Display Format	ENG display } SCI display if both clear.	M,1
42	Grads Mode	GRAD Mode } DEG Mode if both clear.	M,1
43	Radians Mode	RAD Mode } DEG Mode if both clear.	M,1
44	Continuous ON	XEQ ON to set; won't shut off in 10 min.	C
45	System Data Entry	System data entry. Always tests clear.	C
46	Partial Key Sequence	Partial key sequence. Always tests clear.	C
47	Shift	SHIFT Mode. Always tests clear.	C
48	Alpha Mode	ALPHA Mode.	C
49	Low Battery	Low battery.	M
50	Message	Message in display. Always tests clear.	C
51	SST	Single-step. Always tests clear.	C
52	PRGM Mode	Program Mode. Always tests clear.	C
53	I/O	Input/output device is ready (handshake).	NA
54	PSE	Pause in progress. Always tests clear.	NA
55	Printer Existence	Printer is plugged in.	2

*NOTES:
C = Cleared.
M = Maintained by Continuous Memory.
NA = Not applicable.
S = Set.

1 = "Master Clear" clears flag.
2 = Flag 21 is set to match Flag 55 at turn-on.
3 = "Master Clear" sets flag.

INDEX

AUTHOR INDEX:

GLOBAL LABELS USED IN THIS BOOK: ACCHR character order.

-DEC	BL	D-DM	FB	IT	N-A	PRSW	S2	SUP	VK
-INT	BLR	D-F	BD	IV	NC	QE	S3	SV	VP
2V	BI	DC	FFFF	JC	NF	QEQ	SA	SYM	VM
55	BM	DF	FIN	JD	NH	QR	SC	SZ	VMAN
7ENG	BP	DFD	FL	KC	NP	QS	SCE	SZE	VS
7FIX	BR	DIG	FN	LB	NUM?	R-S	SCL	T+S	WRD?
7SCI	BS	DIV	FT	LD	N↑R	R1	SCR	T1	X?Y
=Ø	BV	DN	GCD	LET	N↑S	R2	SCT	T2	XB
?S	BX	DM-D	GD	LG	OUT	R3	SCX	T3	XD
ACOSH	BYX	DR	GN	LINE	OE	RAN	SD	T4	XL
AD	BΣ	DS	GT	LO	P*	RB	SE	T5	XPN
AL	C?	DSPR	HD	LR	P+C	RD	SET	TANH	XR
AM	CD	DSPS	HG	M1	P-S	RDEC	SI	TB	XY
AN	CE	DT	HN	M2	P2	RE	SINH	TC	XYΣ
AP	CFA	DUP	I/	M3	PC	RED	SIZE?	TEM	YN
AR	CFX	DV	IB	M4	PD	RF	SK	TEMP	dB+
ASINH	CHAR	DX	IF	M5	PE	RI	SLD	TITLE	dB-
ATANH	CJ	EB	IG	MA	PERM	RK	SM	TM	Σ3
AV	CLR	EN	IN	MAN	PF	RN	SO	TMP	Σ?
AVN	CLRGX	ER	INB	MANT	PHONE	RNG	SORT	TP	ΣB
AX	CM	ET	INBL	MARY	PM	ROCT	SP	TR	ΣC
B+	CODE	EX	IR	MC	PN	RRM	SR	TX	ΣD
B?	COM	EXP	IS1	MIO	POLY	RS	ST	UD	ΣF
BANR	COMB	F*	IS2	MO	PO	RV	STACK	UR	ΣI
BC	COSH	F+	IS3	MOZ	PR	RΣ	STR	V2	ΣR
BD	CR	F-	IS4	MS	PRL	S-R	STS	VA	ΣRECIP
BE	CU	F-D	IS5	MT	PRM	S?	STX	VB	ΣS
BF	CURT	F/	IS6	MX	PRQE	S1	SU	VF	

TWO-CHARACTER GLOBAL LABELS USED BY H-P: These are the 2-character global labels in the HP-41, its peripherals, and all modules as of August 15, 1981.

*Ø	*9	*L	*b	C*	CO	DL	GC	LS	PV	R↑	UO	W6	X3
*1	*/	*M	*c	C+	CP	DR	GY	MI	Pb	Re	UV	W7	X4
*2	*?	*N	*e	C-	CS	EO	HR	NA	R2	SF	UW	W8	X5
*3	*A	*P	1O	C/	CT	EQ	HT	NE	R=	SH	WØ	W9	X6
*4	*B	*R	AO	C=	CW	E↑	IN	ON	RC	SL	W1	WA	X7
*5	*C	*S	AZ	CF	CZ	FA	JD	PH	RL	ST	W2	WB	X8
*6	*H	*W	BG	CG	DB	FF	L=	PI	RM	TF	W3	XØ	X9
*7	*I	*X	GO	CH	DH	FM	LN	PL	RP	TS	W4	X1	Σ+
*8	*J	*a	BT	CL	DI	FV	LP	PP	RS	UG	W5	X2	Σ-

PPC ROM GLOBAL LABELS NOT IN ABOVE LISTS: PPC is a private users club that supports Hewlett-Packard personal programmable calculators and computers. One of the priceless benefits available only to members is the opportunity to help design, and then purchase, 'limited edition' calculator-related products. One of these has been the PPC ROM—a programmers 8K application module. Many of its routines are in this book; many are not. If you have the extraordinarily good fortune to come into possession of one of these ROMs, you will want to avoid labeling your own programs with the following PPC ROM labels:

+K	2D	BA	CK	CX	EP	FR	HP	L-	ML	NS	PK	RX	Sb
-B	A?	CA	CP	DP	F?	GE	HS	LF	MP	OM	PS	Rb	TN
1K	Ab	CB	CV	E?	FI	HA	IP	MK	NR	PA	RT	SX	XE

HP-41 BYTE TABLE (HEX TABLE)

Columns 0–F: 1-BYTE INSTRUCTIONS (rows 0–7), 2-BYTE (rows 8–C), 3-BYTE (rows D–E), ? (row F). Rows 0–7: POSTFIX DIRECT; rows 8–F: POSTFIX INDIRECT.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0 NULL 00	1 LBL 00 01	2 LBL 01 02	3 LBL 02 03	4 LBL 03 04	5 LBL 04 05	6 LBL 05 06	7 LBL 06 07	8 LBL 07 08	9 LBL 08 09	10 LBL 09 10	11 LBL 10 11	12 LBL 11 12	13 LBL 12 13	14 LBL 13 14	15 LBL 14 15
1	16 0 16	17 1 17	18 2 18	19 3 19	20 4 20	21 5 21	22 6 22	23 7 23	24 8 24	25 9 25	26 . 26	27 EEX 27	28 CHS 28	29 GTO α 29	30 XEQ α 30	31 SPARE 31
2	32 RCL 00 32 SPACE	33 RCL 01 33	34 RCL 02 34	35 RCL 03 35	36 RCL 04 36	37 RCL 05 37	38 RCL 06 38	39 RCL 07 39	40 RCL 08 40	41 RCL 09 41	42 RCL 10 42	43 RCL 11 43	44 RCL 12 44	45 RCL 13 45	46 RCL 14 46	47 RCL 15 47
3	48 STO 00 48	49 STO 01 49	50 STO 02 50	51 STO 03 51	52 STO 04 52	53 STO 05 53	54 STO 06 54	55 STO 07 55	56 STO 08 56	57 STO 09 57	58 STO 10 58	59 STO 11 59	60 STO 12 60	61 STO 13 61	62 STO 14 62	63 STO 15 63
4	64 + 64	65 - 65	66 * 66	67 / 67	68 X<Y? 68	69 X>Y? 69	70 X<=Y? 70	71 Σ+ 71	72 Σ- 72	73 HMS+ 73	74 HMS- 74	75 MOD 75	76 % 76	77 %CH 77	78 P-R 78	79 R-P 79
5	80 LN 80	81 X↑2 81	82 SQRT 82	83 Y↑X 83	84 CHS 84	85 E↑X 85	86 LOG 86	87 10↑X 87	88 E↑X-1 88	89 SIN 89	90 COS 90	91 TAN 91	92 ASIN 92	93 ACOS 93	94 ATAN 94	95 DEC 95
6	96 1/X 96	97 ABS 97	98 FACT 98	99 X≠0? 99	100 X>0? 00 100	101 LN1+X 01 101	102 X<0? A 102	103 X=0? B 103	104 INT C 104	105 FRC D 105	106 D-R E 106	107 R-D F 107	108 HMS G 108	109 HR H 109	110 RND I 110	111 OCT J 111
7	112 CLΣ T	113 X<>Y Z	114 PI Y	115 CLST X	116 R↑ L	117 RDN M [	118 LASTX N \	119 CLX O]	120 X=Y? P ↑	121 X≠Y? Q _	122 SIGN ├ T	123 X<=0? a	124 MEAN b	125 SDEV c	126 AVIEW d	127 CLD e
8	128 DEG 00	129 RAD 01	130 GRAD 02	131 ENTER 03	132 STOP 04	133 RTN 05	134 BEEP 06	135 CLA 07	136 ASHF 08	137 PSE 09	138 CLRG 10	139 AOFF 11	140 AON 12	141 OFF 13	142 PROMPT 14	143 ADV 15
9	144 RCL 16	145 STO 17	146 ST+ 18	147 ST- 19	148 ST* 20	149 ST/ 21	150 ISG 22	151 DSE 23	152 VIEW 24	153 ΣREG 25	154 ASTO 26	155 ARCL 27	156 FIX 28	157 SCI 29	158 ENG 30	159 TONE 31
A	160 XROM 32	161 XROM 33	162 XROM 34	163 XROM 35	164 XROM 36	165 XROM 37	166 XROM 38	167 XROM 39	168 SF 40	169 CF 41	170 FS?C 42	171 FC?C 43	172 FS? 44	173 FC? 45	174 GTO/XEQ 46 IND*	175 SPARE 47
B	176 SPARE 48	177 GTO 00 49	178 GTO 01 50	179 GTO 02 51	180 GTO 03 52	181 GTO 04 53	182 GTO 05 54	183 GTO 06 55	184 GTO 07 56	185 GTO 08 57	186 GTO 09 58	187 GTO 10 59	188 GTO 11 60	189 GTO 12 61	190 GTO 13 62	191 GTO 14 63
C	192 GLOBAL 64	193 GLOBAL 65	194 GLOBAL 66	195 GLOBAL 67	196 GLOBAL 68	197 GLOBAL 69	198 GLOBAL 70	199 GLOBAL 71	200 GLOBAL 72	201 GLOBAL 73	202 GLOBAL 74	203 GLOBAL 75	204 GLOBAL 76	205 GLOBAL 77	206 X<> 78	207 LBL 79
D	208 GTO 80	209 GTO 81	210 GTO 82	211 GTO 83	212 GTO 84	213 GTO 85	214 GTO 86	215 GTO 87	216 GTO 88	217 GTO 89	218 GTO 90	219 GTO 91	220 GTO 92	221 GTO 93	222 GTO 94	223 GTO 95
E	224 XEQ 96	225 XEQ 97	226 XEQ 98	227 XEQ 99	228 XEQ 00 100	229 XEQ 01 101	230 XEQ A 102	231 XEQ B 103	232 XEQ C 104	233 XEQ D 105	234 XEQ E 106	235 XEQ F 107	236 XEQ G 108	237 XEQ H 109	238 XEQ I 110	239 XEQ J 111
F	240 TEXT 0 T	241 TEXT 1 Z	242 TEXT 2 Y	243 TEXT 3 X	244 TEXT 4 L	245 TEXT 5 M [	246 TEXT 6 N \	247 TEXT 7 O]	248 TEXT 8 P ↑	249 TEXT 9 Q _	250 TEXT 10 ├ T	251 TEXT 11 a	252 TEXT 12 b	253 TEXT 13 c	254 TEXT 14 d	255 TEXT 15 e
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

* Byte 174 is GTO IND if the byte following is from the top half of the table (Rows 0-7); it is XEQ IND if the byte following is from the bottom half of the table (Rows 8-F).